FLUID-STRUCTURE SIMULATION OF LARGE MICROMACHINED ULTRASONIC TRANSDUCER ARRAYS

A Thesis
Presented to
The Academic Faculty

By

Bernard Shieh

In Partial Fulfillment
Of the Requirements for the Degree
Doctorate of Philosophy in the
School of Mechanical Engineering

Georgia Institute of Technology

August, 2017

FLUID-STRUCTURE SIMULATION OF LARGE MICROMACHINED ULTRASONIC TRANSDUCER ARRAYS

Approved by:

Dr. Karim Sabra, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. F. Levent Degertekin, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Julien Meaud
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Stanislav Emelianov
School of Biomedical Engineering
Georgia Institute of Technology

Dr. Massimo Ruzzene
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: August 1st, 2017

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

SUMMARY

Acoustic transducer arrays composed of a large number of flexural membranes are found in a diversity of applications such as underwater acoustics, medical therapeutics, and medical imaging. By coupling the vibration of electromechanical layers to the flexural motion of low-impedance membrane structures, these transducers can achieve large bandwidths in immersion without the complexity of matching and backing layers associated with traditional piezoelectric designs. Accurate simulation of array behavior is computationally challenging due to the large number of membranes, complicated by the acoustic interaction between the membranes which has been shown to degrade the frequency and directional characteristics. The acoustic interactions are a result of evanescent standing waves above the fluid-solid interface with resonant behavior related to the geometric layout of the array, the location of the array boundaries, and the individual membrane resonances. Modeling approaches such as finite element analysis (FEA) and boundary element methods (BEM) are computationally difficult due to mesh sizes which become prohibitively large as the number and size of the membranes increase. In this work, we develop a framework for the efficient simulation of fluid-structure coupling for large membrane-type transducer arrays. The simulation is based on a BEM model with improved computational efficiency through the application of a multi-level fast multipole algorithm (ML-FMA). By leveraging truncated multipole expansions for distant membrane pairs, the resulting algorithm improves the theoretical asymptotic space and time complexity from quadratic and cubic time, respectively, to quasilinear time in both cases. We verify the model with FEA and direct BEM solutions. The accurate modeling of large transducer arrays is particularly relevant to emerging technologies such as capacitive (CMUTs) and piezoelectric micromachined ultrasonic transducers (PMUTs). We extend this simulation framework to PMUT devices by numerical computation of the piezoelectrically-induced load using a hybrid FEM/BEM approach. This framework is used to investigate the performance of common CMUT and PMUT designs, as well as a novel foldable large aperture 2-D CMUT array for intracardiac echocardiography.

CHAPTER 1

INTRODUCTION

Every year, cardiovascular disease (CVD) claims the lives of more than 801,000 Americans–nearly 1 out of every 3 deaths in the US [12]. Globally, CVD is the leading cause of death, accounting for more than 17.3 million deaths annually. It is estimated that about 92.1 million American adults are currently experiencing some form of CVD or the after-effects of stroke. Although the annual death rate due to heart disease has decreased in the last decade, risk factors such as obesity, nutrition, and physical inactivity remain alarmingly high.

Treatment of CVD range in intensity from lifestyle changes and medication to cardiac (open heart) surgery. In the last 35 years, interventional procedures (which *intervene* before surgery is necessary), have gained a strong foot-hold as a viable minimally-invasive alternative to open heart surgery. First pioneered in the 1960s for the treatment of coronary stenosis (using a procedure now known as balloon angioplasty), technological advances have led to recent introductions of interventional procedures for heart valve replacement and repair. As an indication of adoption volume, over 1 million diagnostic catheterization procedures [13], around 600,000 coronary angioplasty and stent placement procedures [14], and 12,000 transcatheter aortic valve replacement (TAVR) procedures [15] are performed annually in the US.

At the core of interventional cardiology is the catheter, the slender tube inserted into the body which delivers the necessary treatment and diagnostic tools. During an interventional procedure, visualization of anatomy surrounding the catheter tip is critical for guidance, diagnostics, and monitoring. One of the key visualization technologies available to the interventional cardiologist is intracardiac echocardiography (ICE) which provides detailed images of anatomical structures using ultrasonic waves. In the following section, we provide a brief history and overview of ICE development and its importance to modern medicine.

1.1 Intracardiac echocardiography

Catheter tip ultrasonic probes that could be inserted into the body to interrogate internal structures of the heart were envisioned as early as 1960 [16]. The first devices of its kind, utilizing a single piezoelectric crystal, were capable of performing only simple pulse-echo measurements such as time-of-flight [17] [18]. Even so, early experiments in animals established the safety of such devices

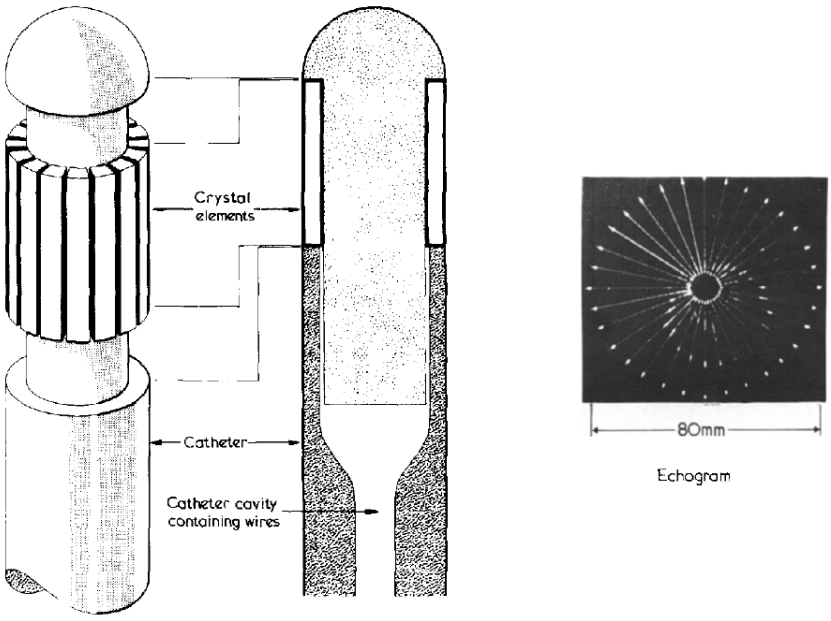


Figure 1: A prototype ICE catheter with piezoelectric elements arranged cylindrically around the catheter tip (left) [1]. An echogram is produced by phasing 8-element subgroups in transmit and receive mode (right).

and their great potential for diagnosis of heart problems. The next two decades saw rapid development in the area, resulting in a variety of advanced working prototypes which would pave the way for clinical use. Notable of these are a single element mechanically rotated probe developed in 1962 which was capable of producing a C-scan image [19], a four-element cylindrical probe in 1969 [20], and a 32-element cylindrical phased array in 1971 [1]. The latter represented the best array and electronics technology available at that time that would fit on the tip of a 9 Fr catheter. The 32 elements of the array, arranged along the cylindrical perimeter of the catheter tip, were phased in subgroups of 8 elements each to produce an echogram with 32 radial lines (see Fig. 1).

Today, ICE has become an invaluable imaging tool for the interventional cardiologist. Modern ICE images produced by phased arrays are real-time grayscale mappings of anatomical features along cross-sectional planes, typically with a 90° field-of-view (FOV) and a maximum 8 - 12 cm depth looking from the right atrium. Systems equipped with Doppler and color flow modes can provide additional visualization of hemodynamic activity. Some examples of common ICE views with Doppler overlays are shown in Fig. 2. Real-time capability combined with good near-field detail elevate ICE images in the catheter lab over other common modalities such as MRI, CT, and fluoroscopy. In the last decade, ICE has superseded transesophageal echocardiography (TEE) as the preferred ultrasonic tool, due in part to improved patient comfort, superior image detail, and a reduction in the level of anesthesia required (local for ICE and general for TEE) [21]. For these reasons, ICE has become an

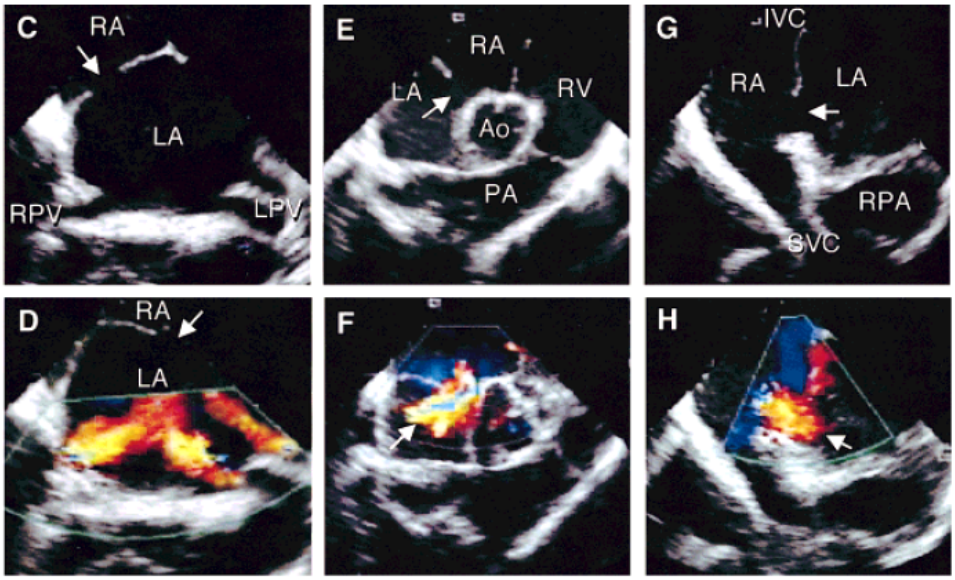


Figure 2: Common ICE views of heart anatomy shown superimposed with Doppler (blood flow) information [2]. LA/RA: left/right atrium, LV/RV: left/right ventricle, LPV/RPV: left/right pulmonary vein, Ao: aorta, IVC/SVC: inferior/superior vena cava.

important tool for guidance and monitoring of complex cardiac procedures.

For example, ICE is routinely used to evaluate defects of the septum (the wall separating the left and right atria) such as atrial septal defects (ASDs) and patent foramen ovale (PFO) and to monitor the deployment of septal occlusion devices used to treat these conditions [2]. Because ICE provides excellent images of the fossa ovalis, access to the left atria via transseptal puncture is often monitored by ICE which can prevent inadvertent puncture of anatomical structures that may be fatal [22]. Furthermore, intravenous injection of microbubbles can be detected in real-time by ICE to verify the position of the sheath in the left atrium.

ICE has also experienced widespread adoption for the monitoring of electrophysiological (EP) procedures due to its ability to detect problems in advance and avoid potential complications. The position of ablation catheters with respect to pertinent cardiac anatomy can be more reliably determined with ICE than with fluoroscopy [23]. Prior to delivery of radiofrequency energy for lesion formation, ICE can be used to confirm adequate electrode contact and stability resulting in improved energy delivery. Visualization of microbubble formation during energy delivery can alert of superheating even before its manifestation as an impedance change, preventing tissue damage.

As the prevalence of ICE in the catheter lab continues to grow, new technological advances are being developed with promising clinical utility. One of the greatest ambitions of ICE development is the realization of real-time volumetric rendering (sometimes referred to as 4-D imaging, with the fourth dimension being time). While volumetric images have been created with ICE catheters before,

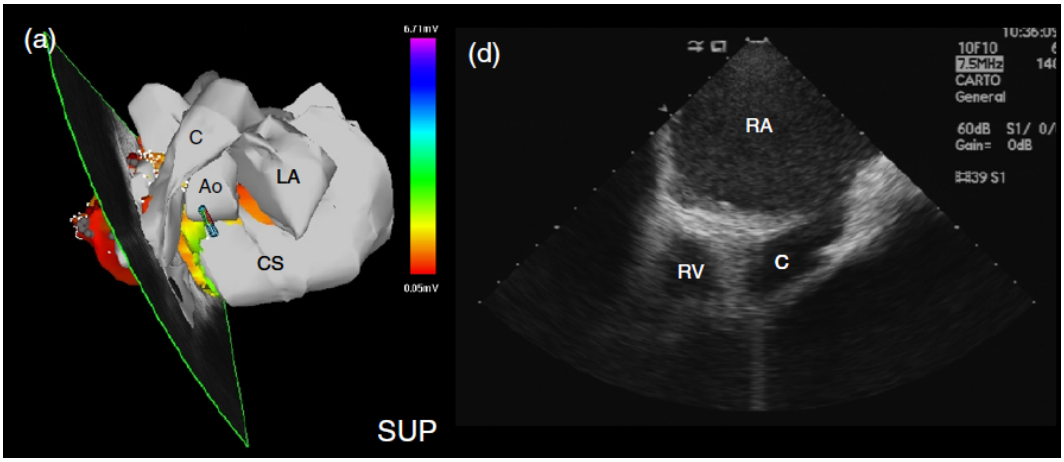


Figure 3: Anatomical mapping systems like CARTO (Biosense Webster) use ICE images to identify and create 3-D shell reconstructions of important anatomical structures [3]. Position and orientation of catheter devices are tracked using magnetic sensors and overlaid onto real-time ICE images.

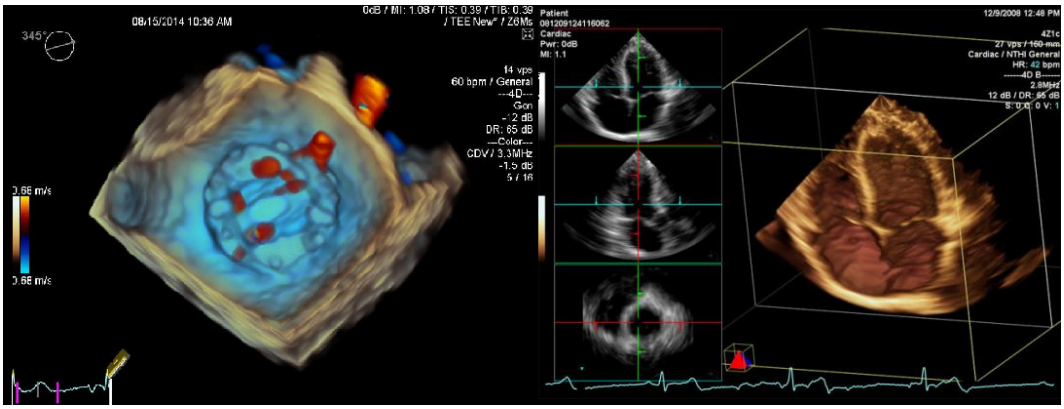


Figure 4: Commercial examples of volumetric images of the heart: TEE from a Siemens Acuson system (left) [4] and ICE using a Siemens Acunav V catheter (right) [5]

these were reconstructions, not real-time images, and involved pullback or rotating mechanisms with ECG gating in order to capture a series of 2-D slice snapshots [16].

1.2 A new generation of acoustic transducers

True 4-D imaging is now available on the latest ultrasound systems for use with probes such as transthoracic echocardiography (TTE) and TEE, but its implementation on a catheter tip device remains a formidable engineering challenge. In particular, the number of array elements will need to increase drastically (from 64 to thousands) and be arranged to sample in two dimensions in order to maintain adequate image quality. Because the size of the cable bundle extending from the transducer is limited by the catheter size, integration with on-tip electronics for preprocessing of real-time data is critical. Other desirable improvements include larger bandwidth for greater frequency agility and resolution, and flexible manufacture and design for potential integration with other catheter devices

such as ablation electrodes and magnetic sensors for electroanatomical mapping (e.g. CARTO system shown in Fig. 3; Biosense Webster; Diamond Bar, CA, USA).

The improvements necessary for 4-D volumetric ICE pose a technical challenge for bulk piezoelectric ceramics–the de facto standard for the construction of medical transducers–for several reasons. Piezoelectric arrays are generally constructed from a larger piezoelectric crystal which is diced and thinned mechanically to achieve the desired element size and resonance frequency. Pieozelectrics operating in the thickness mode have a resonance frequency inversely proportional to its thickness, meaning that for imaging applications around 10 MHz, the material would need to be thinned to around 150 μm. At this scale, mechanical dicing of the crystals becomes a technical challenge due to material brittleness. In addition, bulk piezoelectrics, which generate waves within the material, experience a large acoustic impedance mismatch with water ($Z \approx 20.5$ MRayls for lead metaniobate compared with $Z \approx 1.5$ MRayls for water). In order to efficiently transmit energy into the medium, complex multi-layer muli-material quarter-wavelength matching blocks are required. These matching blocks are cumbersome to construct due to limited material choices, and will ultimately limit the resulting bandwidth of the device.

As a transformative technology, transducers based on microelectromechanical systems (MEMS) are well-suited to tackle these challenges and pave the way forward for real-time volumetric ICE. Micromachined Ultrasonic Transducers (MUTs), such as those actuated by capacitive layers (CMUTs) or by piezoelectric layers (PMUTs), are an emerging technology with a number of inherent advantages over traditional bulk piezoelectrics. By adopting microfabrication techniques from the semiconductor industry, the MUT platform allow for the precise construction of micro-scale mechanical and electrical structures. For ultrasound arrays, this translates into greater design freedom and control over element size, shape, density, and layout. Furthermore, MUTs can be batch-fabricated, with the potential for lower per-device cost and integrated with CMOS electronics using any one of the various techniques that have been demonstrated so far, such as CMUT-in-CMOS [24], interleaved CMUT-CMOS [25] [26], CMUT-on-CMOS [27], and flip-chip bonding [28]. The potential for direct integration with electronics is paramount for an ultrasound transducer since electronics handle critical tasks such as multiplexing, amplification, and complex pulsing and beamformation.

Another important benefit of MUTs is that, unlike bulk piezoelectrics, waves are generated due to the continuity of motion between a membrane surfaces and fluid medium. Fluid-loading therefore plays a dominant role in determining the overall impedance of the structure, meaning that resonance and bandwidth characteristics can be optimized by selection of parameters such as membrane shape,

mass, density, and thickness. Furthermore, with a mechanical impedance naturally matched with water, the need for complex matching layers is obviated.

To date, MUTs have been demonstrated for a diversity of applications. MUTs have been designed for the purposes of rangefinding [29], sound projection [30], and fingerprint identification [31]. In the field of medical sonography, prototype devices for intracardiac echocardiography [32], intravascular ultrasound [33, 34], and photoacoustic imaging [35] provide a glimpse into future imaging platforms based on MUT technology. MUTs are also a unique candidate for super-resolution imaging of biological samples using acoustic time-reversal [36].

1.3 Flextensional transducers, CMUTs, and PMUTs

Modern MUTs are predated by nearly half a century by their larger counterpart, the flextensional transducer. A brief history of the flextensional transducer is given here, followed by an explanation of the operating principles of standard CMUT and PMUT designs.

By coupling the extensional motion of an electromechanical driver to the flexural motion of a thin membrane shell, flextensional acoustic transducers are characterized by their power, efficiency, and desirable broadband response in immersion. The earliest known record of an acoustic transducer based on flexural motion is a patent filed in 1936 by Harvey C. Hayes of the U.S. Naval Research Laboratories (NRL), Washington, D.C. [37] [38]. In the patent, Hayes describes an electromechanical driver encased in an oval metallic shell structure. The driver, consisting of coaxial coils wrapped around a nickel rod, worked on the principle of magnetostriction; by driving the coil with an alternating current, the resulting length-wise vibration of the rod would move two pistons (one at each end of the rod) which were coupled to the enclosing shell structure. In this manner, the relatively high mechanical impedance at the rod would be translated into low impedance on the surface of the shell, with the potential for improved radiation efficiency. At the time, Hayes envisioned that his invention would be used as an air-coupled device for applications as an audible foghorn (in fact, some of his financial support originated from the U.S. Lighthouse Service). Unfortunately, interest in the transducer was overshadowed by competing designs–particularly the acoustic ring transducer which was found to be cheaper, more rugged, and just as efficient–and the idea was abandoned.

With the advent of SONAR and the intense research effort which followed, the idea of a flexural transducer was rediscovered in the late 1950s in the context of a water-coupled device, a purpose for which the design truly excelled. The invention in its reincarnated form, often attributed to William J. Toulis of the Naval Electronics Laboratory (NEL), San Diego, CA, replaced the magnetostrictive drive

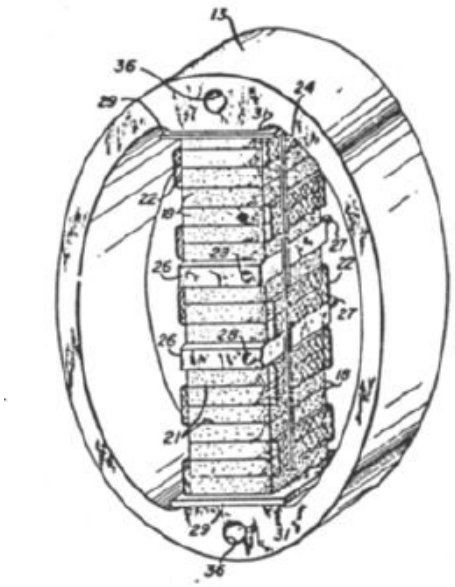


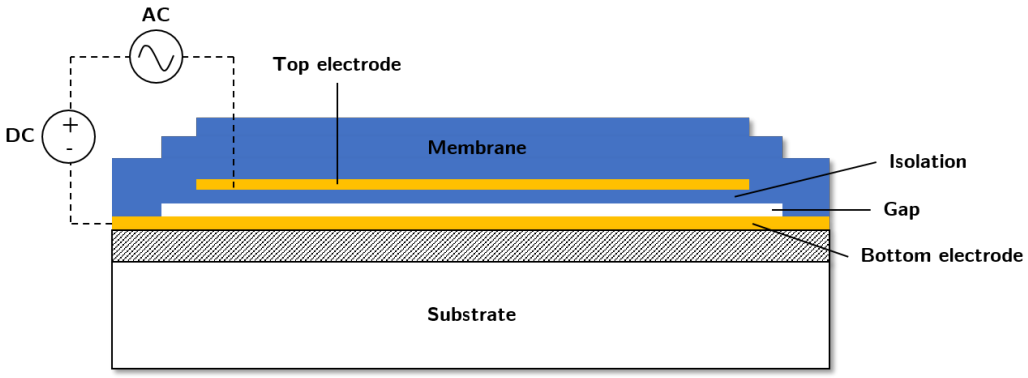
Figure 5: A diagram of the flexural-extensional electro-mechanical transducer from a patent filed in 1966 by William J. Toulis [6].

mechanism with more preferable piezoelectric ceramics (see Fig. 5). Subsequent improvements and variations of this basic design form the basis for a class of transducers referred to now as flextensional (flexural-extensional) transducers after their characteristic ability to convert extensional motion to flexural motion.

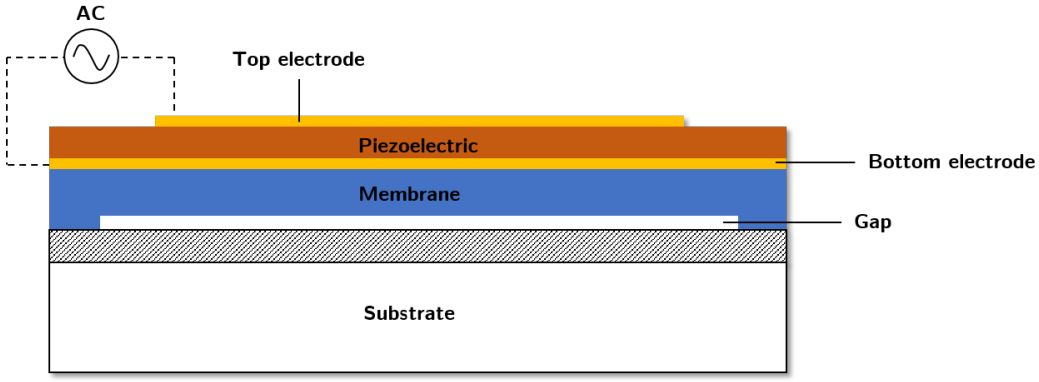
Modern incarnations of the flextensional design (see, for example, the cymbal transducer [39]) are low-frequency, high-power, high-efficiency devices with a strongly-desired broadband response. They have been demonstrated for a variety of applications, including underwater object identification [40], imaging [41], communication [42], transdermal delivery of drugs and insulin [43, 44], and treatment of ulcerations by ultrasonic therapy [45]. Fabrication of arrays for low-frequency operation, with membrane lateral dimensions and thicknesses on the order of millimeters, has been demonstrated with adequate precision using standard, widely-available, low-cost technologies [46,47]. Such a transducer may be manufactured, for example, by attaching a large foil or thin stainless steel membrane layer to a backplate consisting of machined grooves or hollowed cells. Within each cell, individual metallized piezoelectric disks are attached to the membrane layer (with isolating layers as appropriate), forming a unimorph which will deflect due to stresses induced in the disk (for example, see [46]).

With the advent of micromachining, MUTs apply the successful flextensional design to microscale devices. CMUTs and PMUTs alike operate by exciting the flexural motion of thin boundary-clamped membrane structures. They differ, however, in the way the membranes are excited into vibration.

In a CMUT, the membrane is excited by an electrostatic force, similar in manner to a parallel-



(a) CMUT



(b) PMUT

Figure 6: Schematics of standard designs for CMUTs and PMUTs

plate capacitor with a moveable plate. A cross-section of a standard CMUT design is shown in Fig. 6a. The basic structure consists of a bottom electrode, a top electrode embedded in a membrane structure, and a vacuum (or air) gap and isolation layer separating the two. The CMUT is operated in transmit by first applying a static (DC) voltage and then a short pulsed voltage excitation across the electrodes in order to generate acoustic waves. The isolation layer serves to avoid the shorting of the electrodes during full-swing motion and collapse by preventing physical contact between the electrodes. In receive, electronic circuitry measures the voltage across the electrodes as incoming waves cause the membrane to vibrate. Common material choices include gold (Au), aluminum (Al), copper (Cu), silver (Ag), and platinum (Pt) for the electrode layers, and silicon nitride (Si_xN_y) for the membrane structure and isolation layer. Standard fabrication techniques for CMUTs include wafer-bonding [48] and sacrificial release [49].

In a PMUT, the membrane, which consists of one or more piezoelectric layers sandwiched between passive layers, deflects on the principle of multimorph actuation. When exposed to an electric field, a stress mismatch induced between the piezoelectric and passive layers results in bending of the membrane. A basic unimorph PMUT design is shown in Fig. 6b. Common choice for piezoelectric

material include Lead Zirconate Titanate (PZT), Zinc Oxide (ZnO), Aluminum Nitride (AlN), and Polyvinylidene Flouride (PVDF) [50]. Compared to CMUTs, PMUTs have a larger capacitance and a lower impedance, making them less susceptible to parasitic capacitances and relaxing the requirements of the electronics [51]. PMUTs also benefit from operation that is independent of the gap size or DC bias, making them less sensitive to process variations that may exist over the array. They can be fabricated using similar techniques to CMUTs (wafer bonding, sacrificial release etc.).

1.4 Objectives of this work

Driven by the growing relevance of MUTS for applications in medical imaging, this research is concerned with the development of practical tools for the simulation of large MUT arrays. It is envisioned that accurate and efficient simulation tools will aid in the design, optimization, and general understanding of these arrays. The simulations developed here aim to capture the pertinent physics involved in the imaging process, including the electromechanical behavior of the membranes, the fluid-structure interactions on the transducer surface, and wave propagation and backscatter from representative phantoms. A large emphasis is placed on the development of computationally scalable models which can handle the thousands of membranes expected of realistic imaging arrays.

This work is detailed in the following four chapters. In Chapter 2, a simulation framework is established for the prediction of fluid-structure coupling for membrane-type transducer arrays with computational acceleration using a multi-level fast multipole algorithm. In Chapter 3, the simulation framework is extended to PMUT arrays using a hybrid boundary element method approach. In Chapter 4, examples of large CMUT and PMUT array simulations are provided. In Chapter 5, imaging performance of a novel foldable array for intracardiac echocardiography is explored. Finally, in Chapter 6, concluding remarks and future work are discussed.

CHAPTER 2

EFFICIENT SIMULATION OF FLUID-STRUCTURE COUPLING FOR LARGE ARRAYS

2.1 Background and motivation

It has been well-established that membrane-type arrays are susceptible to mechanical cross-talk which can negatively affect their performance. Early experimental characterization of CMUT arrays determined these effects to be rooted in the complex interaction between membrane vibration and various wave phenomena. In [52], radiation pattern and cross-talk measurements of a 1-D CMUT array indicated the presence of zeroth-order symmetric and anti-symmetric Lamb waves supported by the silicon substrate. Stoneley-type waves at the solid-fluid interface were also determined to be an important source of cross-coupling. In [53], the authors hypothesized that acoustic interaction through the fluid was a major source of cross-coupling. Using optical interferometry of a six-element CMUT array, it was found that element displacement was unaffected by discontinuities and deep cuts in the substrate, signifiying that fluid-born waves, and not Stoneley-type waves, were the important mechanism of cross-coupling.

Cross-talk phenomena continued to be studied using both simulation and experimental measurement. In [54], through finite element method (FEM) simulation and laser interferometry of a 1-D CMUT array, it was determined that the primary cross-talk mechanism was so-called *dispersive guided modes*, with smaller contributions (more than 20 dB below) from Lamb waves and Stoneley-type waves. These dispersive guided modes, which travel along the fluid-solid interface, are supported by the resonance of the membranes and are therefore dependent on the individual membrane dynamics and the geometric layout of the array. A similar conclusion was reached with 3-D modeling of CMUTs based on a periodic finite element method/boundary element method (FEM/BEM) simulation [55]. Dispersive guided modes of CMUT arrays have since been studied in the context of acoustic metamaterials using 1-D and 2-D modal analysis [56].

Unlike bulk piezoelectric transducers, it is clear that the design and optimization of membrane-type arrays cannot be performed without consideration of the fluid-structure interaction problem. A rapid iterative design process will necessitate the efficient simulation of these physics. However, to date, the simulation of membrane-type arrays with several thousand membranes has not been achieved without significant concessions. For example, modeling full arrays using finite element method (FEM) software requires a considerable (and often prohibitive) computational effort. These

models may rely on one or more symmetry conditions in order to reduce the problem size (see for example [54]). Periodic boundary conditions have also been employed to simulate unbounded periodic arrays [55, 57, 58], but these are not representative of realistic designs.

To tackle simulation of large arrays, some authors have proposed more efficient semi-analytical models. In this approach, the effect of dispersive guided modes are captured through approximate expressions for the mutual radiation impedance between membrane pairs. For example, infinite series expressions for the mutual radiation impedance can be derived for flexural disks in a rigid, plane baffle for various edge boundary conditions, as in [59]. These expressions have been adopted to study cross-talk in CMUT arrays [60–62] and PMUT arrays [63]. An alternative but similar approach based on eigenmodes derived from a simple plate theory is used in [64]. Two significant assumptions are made in these works which limit their generality: (1) membranes are limited to circular disks which may not always be preferrable (e.g., square and hexagonal membranes will have a larger surface area fill-factor), and (2) membranes are restricted to radially-symmetric motion which ignores contribution of anti-symmetric modes. In [60], it was determined that anti-symmetric modes may contribute to membrane motion depending on the frequency and relative position of the membranes.

To continue to improve the design of large membrane-type arrays for potential commercial applications, a simulation tool is needed which is accurate, computationally efficient, and sufficiently general to cover cases of practical interest. A numerical model based on the boundary element method (BEM) [65, 66] is a potential candidate which may provide both the efficiency and flexibility desired. Recently, this BEM approach has been verified with FEM software (COMSOL Multiphysics) and experimentally for a dual-ring CMUT array with good agreement [67]. Unfortunately, the BEM model suffers from unsatisfactory memory and run-time scaling due to its dependence on a fully-populated mutual impedance matrix, limiting its utility for the simulation of large arrays. For example, an array with 370 membranes (around 30,000 nodes) would require a matrix with 450,000 entries (7 GiB of storage) and would take about 5 hours to solve at 1 GFlops.

In this chapter, we address the scaling problem of the BEM model directly through the application of a fast multipole algorithm (FMA). This novel algorithm was invented in 1987 by Rokhlin and Greengard [68] for the rapid evaluation of electric and gravitational fields involving a large number of particles. Since then, it has been applied in various forms to problems in electromagnetics [69], molecular dynamics [70], and acoustics [71–73], amongst others. An FMA-accelerated BEM model for membrane-type arrays can handle several thousand membranes with reasonable computational resources while retaining the flexibility necessary for simulating arrays of practical interest. Its

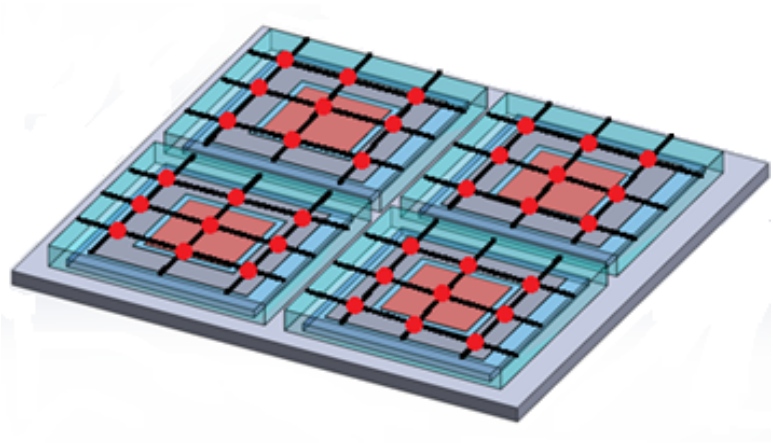


Figure 7: Two-dimensional surface mesh for BEM simulation.

distinguishing features include: simulation of arrays of finite size; simulation of arbitrary array layouts, i.e. no symmetry or periodicity required; support for arbitrary membrane shapes, e.g. circular and rectangular; simultaneous simulation of different membrane types, e.g. different gap sizes, membrane thicknesses, isolation thicknesses; and the ability to phase membranes arbitrarily.

The structure of this chapter is as follows. The BEM model for membrane-type arrays and the underlying equation which must be solved are reviewed. The theory of the fast multipole algorithm and the operations of a multi-level scheme are explained in detail. Practical matters of preconditioning and controlling error are addressed. Further optimizations of the algorithm such as Fourier-based interpolation and filtering, and pre-caching are described. The model is validated against solutions from direct BEM and FEM simulations. Computation and memory usage are compared. Finally, discussion and concluding remarks are given.

2.2 A boundary element method for membrane-type arrays

A numerical model of membrane motion based on the boundary element method strikes a good balance between complexity and efficiency, achieved by leveraging boundary integral equations for acoustics. Such a model has been demonstrated extensively in previous works for the simulation of CMUTs [65, 66, 74–76]. In BEM, in contrast with 3-D FEM, the problem is compressed from three dimensions (x, y, z) to two (x, y), avoiding the need to mesh the interior structures and the fluid, but limiting the cross-talk mechanism to fluid-born waves, i.e. the dispersive guided modes, since the substrate dynamics are excluded. A surface mesh is defined (see Fig. 7), where equilibrium for the mesh nodes are described in terms of lumped mass, damping, stiffness, and radiation impedance.

This is expressed by the following linear system of angular frequency ω

$$(-\omega^2\mathbf{M} + i\omega\mathbf{C} + \mathbf{K} + i\omega\mathbf{Z}_{rad})\vec{w} = \vec{p}_{act} \tag{1}$$

It is sometimes useful to group these matrices into mechanical and radiation components, such that

$$\begin{aligned}(\mathbf{G}_{mech} + \mathbf{G}_{rad})\vec{w} = \mathbf{G}\vec{w} = \vec{p}_{act} \\ \mathbf{G}_{mech} = -\omega^2\mathbf{M} + i\omega\mathbf{C} + \mathbf{K} \\ \mathbf{G}_{rad} = i\omega\mathbf{Z}_{rad}(\omega)\end{aligned} \tag{2}$$

For a mesh with I nodes, each of these matrices will have dimensions $I \times I$. The composition of each matrix is described in the following.

$\mathbf{M}$ is a mass matrix with diagonal entries representing the mass per unit area of each node, i.e. for the n-th node, $M_{nn} = \rho_1 h_1 + \rho_2 h_2 + \cdot + \rho_L h_L$ in the case of an L-layer composite plate, where ρ_l is the density and h_l is the thickness of each layer.

$$\mathbf{M} = \begin{bmatrix} M_{11} & & & \\ & M_{22} & & \\ & & \ddots & \\ & & & M_{II} \end{bmatrix} \tag{3}$$

$\mathbf{C}$ is a damping matrix, where we have chosen to employ a simple nodal damping model such that the diagonal entries are $C_{nn} = \beta$ for some damping coefficient β.

$$\mathbf{C} = \begin{bmatrix} \beta & & & \\ & \beta & & \\ & & \ddots & \\ & & & \beta \end{bmatrix} \tag{4}$$

Coupled behavior of the nodes arises from the two remaining matrices. The $\mathbf{K}$ matrix captures the mutual stiffness between nodes of the membrane. For thin membranes, $\mathbf{K}$ can be derived by finite difference approximation of the equation of motion obtained from classical thin-plate theory.

$$\mathbf{K} \approx -D_{comp} \cdot \nabla^4 w \tag{5}$$

where D_{comp} is the flexural rigidity of the composite plate [77], and ∇^4 is the biharmonic operator.

In cartesian coordinates this is

$$\nabla^4 = \frac{\partial^4}{\partial x^4} + 2\frac{\partial^2}{\partial x \partial y} + \frac{\partial^4}{\partial y^4} \tag{6}$$

For example, the fourth derivative at a node x_n can approximated by the following second-order finite difference

$$\frac{\partial^4 f_n}{\partial x^4} \approx \frac{1}{\Delta x^4} \begin{bmatrix} 1 \\ -4 \\ 6 \\ -4 \\ 1 \end{bmatrix} \begin{bmatrix} f_{n-2} \\ f_{n-1} \\ f_n \\ f_{n+1} \\ f_{n+2} \end{bmatrix} + O(\Delta x^4) \tag{7}$$

where Δx is the node spacing and the shorthand notation $f_n = f(x_n)$ is used. Decomposition of ∇^4 into finite differences will yield a stencil relating the displacement of a node with the displacement of its neighbors. A second-order stencil for (6) on a regular node grid is constructed as follows

$$\nabla^4 \approx \frac{1}{\Delta x^4} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 6 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \frac{1}{\Delta y^4} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -4 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & -4 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} +$$

$$\frac{1}{\Delta x^2 \Delta y^2} Conv2d \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -1/12 & 4/3 & -5/2 & 4/3 & -1/12 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & -1/12 & 0 & 0 \\ 0 & 0 & 4/3 & 0 & 0 \\ 0 & 0 & -5/2 & 0 & 0 \\ 0 & 0 & 4/3 & 0 & 0 \\ 0 & 0 & -1/12 & 0 & 0 \end{bmatrix} \right\} \tag{8}$$

where $Conv2d$ is the 2-D convolution operator with only the central portion retained.

Note that **K** couples the nodes of a membrane with other nodes of the *same* membrane; in this formulation, membranes are not coupled structurally with the other membranes of the array. It

follows that $\mathbf{K}$ will be block-diagonal with blocks $\mathbf{K_1}, \mathbf{K_2}, \ldots, \mathbf{K_I}$ for an array with I total membranes.

$$\mathbf{K} = \begin{bmatrix} \mathbf{K_1} & & & \\ & \mathbf{K_2} & & \\ & & \ddots & \\ & & & \mathbf{K_I} \end{bmatrix} \tag{9}$$

Furthermore, if the membranes are geometrically identical, which is typically the case, then $\mathbf{K_1} = \mathbf{K_2} = \cdots = \mathbf{K_N}$

For membrane geometries which are not accurately described by thin-plate theory, e.g. thick, mass-loaded, stepped, or curved membranes, $\mathbf{K}$ can be derived numerically from FEM simulation. The following procedure was first proposed by Zahorian et. al. [27]. First, a model of the membrane is built in FEM software. Static deformation of the membrane is calculated under a 1 Pa load applied individually to each node of the membrane, where care is taken to ensure that the BEM and FEM nodes correspond to the same locations. For each applied load, the displacements calculated on the node grid of N total nodes are used to form the columns of the compliance matrix $\mathbf{K}^{-1}$. That is,

$$\mathbf{K_1} = \begin{bmatrix} \vec{w}_1 & \vec{w}_2 & \cdots & \vec{w}_N \\ & & & \end{bmatrix}^{-1} \tag{10}$$

where $\vec{w}_n$ is the displacement for loading of the n-th node.

The radiation impedance matrix $\mathbf{Z}_{rad}$ captures the acoustic cross-coupling between nodes. The nodes are modeled as baffled simple sources of surface area a_n which radiate according to the three-dimensional free space Green's function. A nodes effect on itself (its self-impedance) is approximated by that for a baffled circular piston radiator with equivalent surface area [78]. The entries of $\mathbf{Z}_{rad}$ are therefore

$$Z_{rad}^{mn} = \begin{cases} i\omega\rho a_n \dfrac{e^{-ik|\vec{r}_m - \vec{r}_n|}}{2\pi|\vec{r}_m - \vec{r}_n|}, & \text{for } m \neq n \\ \rho c \left[\dfrac{1}{2\pi} k^2 a_n + i \dfrac{8}{3\pi} k \left(\dfrac{a_n}{\pi} \right)^{1/2} \right] & \text{for } m = n \end{cases} \tag{11}$$

where ρ is the fluid density, c is the fluid sound speed, k is the wavenumber, and $\vec{r}_m$ and $\vec{r}_n$ are the vectors pointing to the m-th and n-th node, respectively.

It is useful conceptually to partition $\mathbf{Z}_{rad}$ into block matrices representing membrane-to-membrane interactions and to group the self and mutual blocks. We denote a membrane's self-impedance (re-

stricted to nodes belonging to the same membrane) as $\mathbf{Z}_{self,ii}$, and a membrane's mutual impedance with another membrane as $\mathbf{Z}_{ij}$. Then,

$$\mathbf{Z}_{rad} = \mathbf{Z}_{self} + \mathbf{Z}_{mutual}$$

$$\mathbf{Z}_{rad} = \begin{bmatrix} \mathbf{K}_{self,11} & & & \\ & \mathbf{K}_{self,22} & & \\ & & \ddots & \\ & & & \mathbf{K}_{self,ll} \end{bmatrix} + \begin{bmatrix} & \mathbf{K}_{12} & \dots & \mathbf{K}_{1l} \\ \mathbf{K}_{21} & & & \vdots \\ \vdots & & \ddots & \mathbf{K}_{(l-1)l} \\ \mathbf{K}_{l1} & \dots & \mathbf{K}_{l(l-1)} & \end{bmatrix} \tag{12}$$

Finally, the vector $\vec{p}_{act}$ is a placeholder for the normal load applied by the actuating mechanism. The source of the load will depend on the particular device design, e.g. bending moments induced by piezoelectric layers in the case of PMUTs, and electrostatic force between capacitive layers for CMUT devices. For CMUTs, a useful approximation is to consider small harmonic motion of the membrane about a large static deflection. The loading in this case is linear and related to the applied AC voltages $\mathbf{v}$ by the transformer ratios $\vec{\eta}$ and to the displacement $\vec{w}$ by a spring-softening matrix $\mathbf{K}_{ss}$ [65], such that

$$\vec{p}_{act} = (\mathbf{I}\vec{\eta})\vec{v} + \mathbf{K}_{ss}\vec{w} \tag{13}$$

where $\mathbf{I}$ is the identity matrix.

The numerical solution to (54) can be found simply by calculating $\mathbf{G}^{-1}$, or more efficiently, using direct solvers based on the QR or LU decomposition of $\mathbf{G}$. However, as the scale of the problem extends beyond 30,000 nodes, the global nature of boundary element models ($\mathbf{Z}_{rad}$ is a symmetric, fully-populated matrix) makes the direct approach unviable. The fast multipole algorithm is a solution to the poor scaling of BEM by speeding up the calculation of the matrix-vector product $\mathbf{G}\vec{w}$.

2.3 Fast multipole algorithm

The overarching goal of a fast multipole algorithm (of which there are many sub-types and variations) is to accelerate the calculation of matrix-vector products of certain structured dense matrices in a *matrix-free* manner, i.e. without explicit storage of the matrix itself. The calculation, expedited through the use of truncated multipole expansions of the relevant Green's function, is approximate, but with controllable error that can be adjusted up to machine precision. When paired with fast

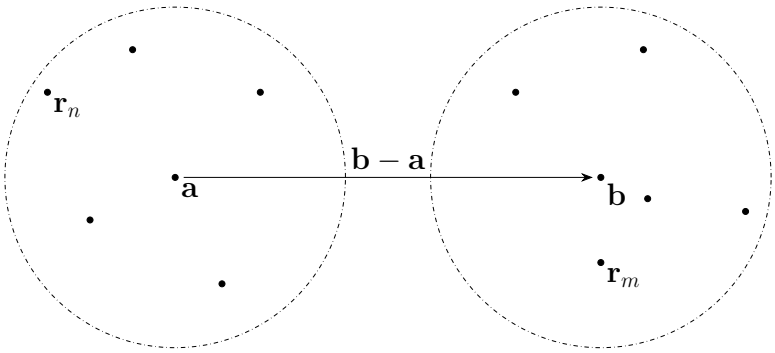


Figure 8: Geometry of the fundamental pressure evaluation problem in the fast multipole algorithm.

iterative forward solvers, e.g. generalized minimal residual (GMRES) [79] and biconjugate gradient (BiCG) [80], it becomes possible to solve linear systems with $O(NlogN)$ complexity in both storage and run-time [81] compared with the $O(\frac{2}{3}N^3)$ complexity of a standard LU solver [80]. Algebraically, the FMA approach can be considered as a type of hierarchical low-rank matrix approximation, although these approximations are not explicitly stored due to its matrix-free property [82].

In this section, we describe the adoption of a fast multipole algorithm for the acceleration of our specific problem. Here, the matrix-vector product of interest is the calculation of the acoustic impedance $i\omega\mathbf{Z}_{rad}\vec{w}$ which represents the pair-wise pressure exerted on each node by all the other nodes. The algorithm consists of multiple interconnected parts, each of which must be properly optimized for a successful implementation. We draw heavily from a large body of literature on the general theory of FMAs and its multiple parts [68, 83–87].

2.3.1 Fundamental pressure evaluation in the fast multipole algorithm

Consider the evaluation of pressure at the *m-th* node located at $\vec{r}_m$ due to the aggregate effect of all the nodes, described mathematically by the action of the matrix-vector product $\mathbf{G}_{rad}\vec{w}$

$$p_m = i\omega\rho \sum_{n\neq m} q_n \frac{e^{-ik|\vec{r}_m-\vec{r}_n|}}{2\pi|\vec{r}_m-\vec{r}_n|} + \rho c q_m \left[\frac{1}{2\pi}(ka_m)^2 + i\frac{8}{3\pi}ka_m\left(\frac{a_m}{\pi}\right)^{1/2}\right] \tag{14}$$

where $q_n = i\omega a_n w_n$ is the complex source strength of the *n-th* node with location $\vec{r}_n$ under a baffled condition. The principle idea behind the fast multipole algorithm is to approximate the fields from distant nodes with truncated multipole expansions. Nodes are collected into clusters based on proximity, and cluster-to-cluster interactions replace node-to-node interactions whenever the clusters are sufficiently separated. The Green's function in (14) is replaced with an expansion derived from a combination of the Gegenbauer addition theorem (10.1.45/46 in [88]) with a plane wave expansion

(refer to [83] for a detailed derivation)

$$\frac{e^{-ik|\vec{x}+\vec{y}|}}{|\vec{x}+\vec{y}|} = \frac{ik}{4\pi}\sum_{l=0}^{\infty}(2l+1)i^l h_l^{(2)}(kx)\int_{S_1} e^{iky\cdot\hat{s}}P_l(\hat{x}\cdot\hat{s})\,dS \tag{15}$$

where $\vec{x}$ and $\vec{y}$ are arbitrary vectors with $x > y$, $h_l^{(2)}(z)$ is the spherical Hankel function of the second kind, and $P_l(z)$ are the Legendre polynomials. The unit vectors $\hat{s}$ are angles of the unit sphere S_1, defined in spherical coordinates by the azimuth angle θ and polar angle ϕ, or in Cartesian coordinates by $\langle cos\theta sin\phi, sin\theta sin\phi, cos\phi\rangle$. The integration is defined over the surface of S_1, where $dS = sin\theta d\theta d\phi$. To understand how this expansion can be used for the evaluation of pressure, let $\vec{x} = \vec{b} - \vec{a}$ and $\vec{y} = \vec{r}_m - \vec{b} + \vec{a} - \vec{r}_n$ for locations $\vec{a}$ and $\vec{b}$, maintaining the stipulation that $|\vec{b} - \vec{a}| > |\vec{r}_m - \vec{b} + \vec{a} - \vec{r}_n|$. Substitution into (15) yields

$$\frac{e^{-ik|\vec{r}_m-\vec{r}_n|}}{|\vec{r}_m-\vec{r}_n|} = \lim_{L\to\infty}\frac{ik}{4\pi}\int_{S_1} e^{ik(\vec{r}_m-\vec{b})\cdot\hat{s}}T_L(\hat{s},\vec{b}-\vec{a})e^{ik(\vec{a}-\vec{r}_n)\cdot\hat{s}}\,dS \tag{16}$$

where we have interchanged the summation and integration, and defined a truncated *translation operator* T_L

$$T_L(\hat{s},\vec{b}-\vec{a}) = \sum_{l=0}^{L}(2l+1)i^l h_l^{(2)}(k|\vec{b}-\vec{a}|)P_l\left(\hat{s}\cdot\frac{\vec{b}-\vec{a}}{|\vec{b}-\vec{a}|}\right) \tag{17}$$

We refer to $\vec{a}$ and $\vec{b}$ as the source and evaluation cluster centers, respectively, for clusters of nodes located within some expansion radius about each location (see Fig. 8). Note that T_L depends only on the vector separating the cluster centers and not on the spatial distribution or the monopole strengths of the nodes within the clusters. This mathematical separation plays a critical role in the resulting computational speed-up of the algorithm.

The fundamental cluster-to-cluster pressure evaluation occurs in three steps. First, the nodes in the source cluster are *aggregated* about $\vec{a}$ by calculation of their *far-field signature*

$$F_{\vec{a}}(\hat{s}) = \sum_n q_n e^{ik(\vec{a}-\vec{r}_n)\cdot\hat{s}} \tag{18}$$

where q_n and r_n are the source strength and position of the *n-th* node, respectively, and $F_{\vec{a}}(\hat{s})$ is a function on the unit sphere. Second, the far-field signature is multiplied by the translation operator,

converting it into a *near-field signature* and shifting the cluster center from $\vec{a}$ to $\vec{b}$

$$N_{\vec{b},L}(\hat{s}) = T_L(\hat{s}, \vec{b} - \vec{a}) F_{\vec{a}}(\hat{s}) \tag{19}$$

$N_{\vec{b},L}(\hat{s})$ is likewise a function on the unit sphere. Finally, the near-field signature is *disaggregated* to determine the pressure at the *m-th* node in the evaluation cluster by carrying out the integration

$$p_m = \lim_{L\to\infty} -\rho c \frac{k^2}{8\pi^2} \int_{S_1} e^{ik(\vec{r}_m - \vec{b})\cdot\hat{s}} N_{\vec{b},L}(\hat{s})\, dS \tag{20}$$

The numerical implementation of this procedure requires two approximations: truncation of the translation operator at the *L-th* term and evaluation of the integral using numerical quadrature over the unit sphere. A trapezoidal quadrature rule with a uniform sampling of the sphere is a straight-forward way to handle the numerical integration. In this case, the weights w_s are constant and depend on the total number of sampling points N_θ and N_ϕ in each direction.

$$p_m \approx -\rho c \frac{k^2}{8\pi^2} \sum_{\hat{s}} w_s e^{ik(\vec{r}_m - \vec{b})\cdot\hat{s}} N_{\vec{b},L}(\hat{s}) \tag{21}$$

$$w_s = \frac{2\pi}{N_\theta} \frac{\pi}{N_\phi} \tag{22}$$

2.3.2 Multi-level adaptive scheme

To achieve an optimal algorithm, the cluster-to-cluster pressure evaluation described above is paired with a scheme to adaptively adjust the precision of the expansion. Larger clusters (in terms of expansion radius) reduce the total number of interactions necessary but require more terms of T_L and a finer quadrature sampling in order to adequately sample the field. By scaling the cluster size with the distance of the interaction, a balance is struck between the number of interactions and the computational cost per interaction.

A multi-level algorithm introduces a hierarchical tree structure to manage such a scheme in the form of a quadtree (specific to problems in two dimensions). A quadtree is composed of multiple levels (refer to Fig. 9). The top level $\mathcal{L}_0$ (the trunk) contains a single bounding box which encloses the entire problem domain. Subsequent levels are formed by repeated bisection of the bounding box in both dimensions, up to a desired maximum level $\mathcal{L}_{max}$ (the leaves). We refer to the bisected box as the *parent* and the four resulting boxes as the *children*. It follows that the trunk will have no parent and the leaves will not have any children.

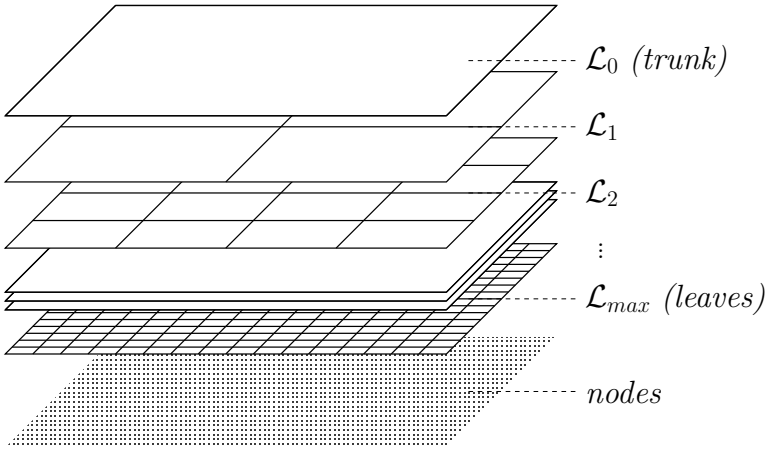


Figure 9: Multi-level quadtree structure which organizes nodes into boxes of decreasing size.

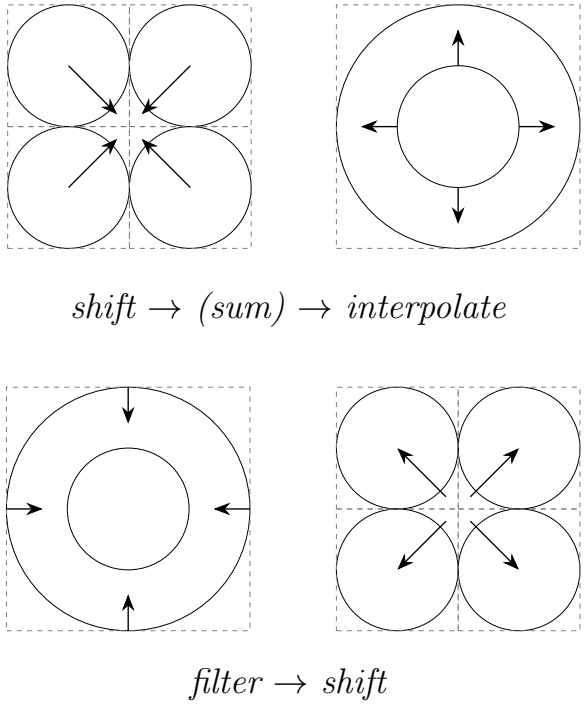


Figure 10: Diagram illustrating the shift, interpolate, and filter operations of the multi-level algorithm. In the upward pass, parent boxes acquire far-field signatures from their children using a shift-sum-interpolate operation. In the downward pass, child boxes inherit near-field signatures from their parents using a filter-shift operation.

At the maximum level, multipole expansions are calculated for the node clusters within each box in the form of far-field signatures. Rather than calculate the signatures for the boxes in the remaining levels, the signatures of parent boxes are acquired from their children through an efficient process. The child signatures, which are not valid expansions for the parent box, must first be manipulated through the application of several operations. These are illustrated in Fig. 10.

A *shift* operation is defined which relocates a signature's center from one location to another. Shifting the center of a far-field (or near-field) signature from **c** to **d** is a simple multiplication with

a complex exponential

$$F_{\mathbf{d}}(\hat{s}) = e^{ik(\mathbf{d}-\mathbf{c})\cdot\hat{s}} F_{\mathbf{c}}(\hat{s}) \tag{23}$$

For a given parent box, the signature of each child box is shifted from its geometric center to the center of the parent box and then summed together.

Next, the signature's expansion radius must be enlarged to account for the increased detail required of the parent box. Each signature, as functions on the unit sphere, is *interpolated* onto the sampling points of a finer quadrature rule with order selected appropriately for the size of the parent box. Much has been written on the optimal method for performing this interpolation process (and the reciprocal *filtering* process), the choice of which will be tied intimately with the selected quadrature scheme. We opt for a Fourier-based method [84] [85] for its ease of implementation and dependence on widely-available Fast Fourier Transform (FFT) routines. The reader should be aware of the alternatives, which include Lagrange polynomials [86], spherical filtering [87], and many others. After interpolation, the resulting signature is a valid expansion of the field from the cluster containing all the nodes within the parent box. In a similar fashion, child boxes can inherit signatures from their parent boxes, a process which is used to efficiently transfer near-field signatures down the quadtree. When moving from a parent to a child, the signature is shifted and then *filtered* onto the sampling points of a coarser quadrature rule.

The adaptive nature of the calculation is realized discretely by classifying all box-to-box interactions on a given level into three categories. These classifications are illustrated in Fig. 11. The closest interactions are those from within the box and from neighboring boxes, i.e. those sharing a border or a vertex. A given box may therefore have a maximum of eight neighbors. These interactions are always computed directly using (14). The intermediate category includes interactions from non-touching neighbors (*ntn*), defined as the children of the neighbors of the parent box, excluding the children which are also neighbors. A given box may have up to 27 non-touching neighbors. Interactions in this category are computed with the FMA using a cluster size equal to the box size for the level. Finally, the interactions with the remaining boxes in the level are categorized as far-away and handled by levels above using larger clusters. Defined in this way, the three categories–neighbors, non-touching neighbors, and far-away–are mutually exclusive. As a matter of implementation, each box of the quadtree should maintain lists identifying its neighboring boxes and non-touching neighbors on the same level.

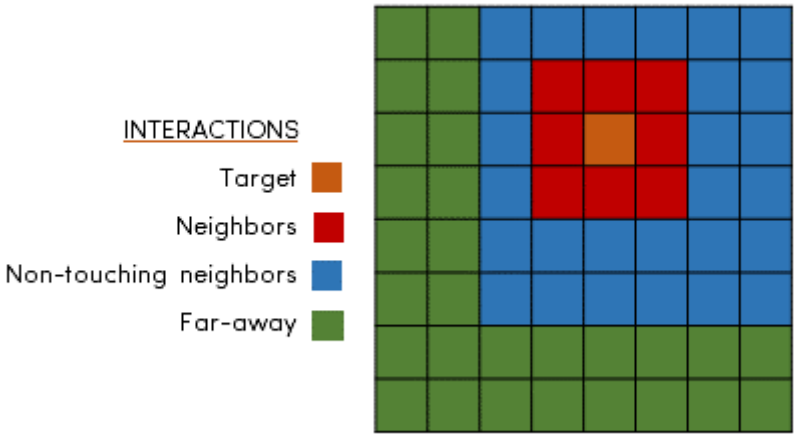


Figure 11: Interaction classifications for the multi-level fast multipole algorithm.

2.3.3 Pressure evaluation using a tree traversal

Recall the ultimate purpose of the algorithm: to replace the costly matrix-vector product $\mathbf{G}_{rad}\mathbf{w}$ with a more efficient calculation. For a given set of node displacements $\mathbf{w}$ (or monopole strengths $\mathbf{q}$), the evaluation of acoustic pressure at every node is carried out by a single traversal of the quadtree. The traversal can be divided into three parts: an upward pass, a downward pass, and an evaluation; we describe these parts explicitly here.

Upward pass:

1. At the bottommost level $\mathcal{L}_{max}$, the far-field signatures are calculated using (18) for every box

$$F_{\vec{a}}(\hat{s}) = \sum_{\text{node } n} q_n e^{ik(\vec{a}-\vec{r}_n)\cdot\hat{s}} \text{ for } box \in \mathcal{L}_{max}$$

   where $\vec{a}$ is the box center, and q_n and $\vec{r}_n$ are the strength and position, respectively, of the nodes in the box.

2. Moving up one level, the boxes in this level acquire the far-field signatures from their children by a shift-sum-interpolate operation

$$F_{\vec{a}}(\hat{s}) = Interp\left[\sum_{child\ j} e^{ik(\vec{a}-\vec{a}_j)\cdot\hat{s}} F_{\vec{a}_j}(\hat{s})\right] \text{ for } box \in \mathcal{L}$$

   where $\vec{a}$ is the box center and $F_{\vec{a}_j}(\hat{s})$ is the far-field signature of the *j-th* child about its center $\vec{a}_j$.

3. Step 2 is repeated for the remaining levels up to and including $\mathcal{L}_2$. At the conclusion of the

upward pass, every box in the levels $\mathcal{L}_{max}, \ldots, \mathcal{L}_2$ will have a far-field signature.

Downward pass:

4. Beginning with $\mathcal{L}_2$, each box in the level acquires the far-field signatures from its non-touching neighbors by translating them one-by-one (converting them to near-field signatures in the process) and summing them together

$$N_{\vec{b},L}(\hat{s}) = \sum_{ntnj} T_L(\hat{s}, \vec{b} - \vec{a}_j) F_{\vec{a}_j}(\hat{s}) \text{ for } box \in \mathcal{L}$$

where $\vec{b}$ is the box center and $F_{\vec{a}_j}(\hat{s})$ is the far-field signature of the *j-th* non-touching neighbor about its center $\vec{a}_j$.

5. Moving down one level, each box inherits the near-field signatures from their parent by a filter-shift operation

$$N_{\vec{b}_j,L}(\hat{s}) = Filter\left[e^{ik(\vec{b}_j - \vec{b})\cdot\hat{s}} N_{\vec{b}}(\hat{s})\right] \text{ for } box \in \mathcal{L}$$

where $\vec{b}_j$ is the center of the *j-th* child box and $N_{\vec{b}}(\hat{s})$ is the near-field signature of the parent about its center $\vec{b}$.

6. Steps 4 and 5 are repeated for the remaining levels up to and including $\mathcal{L}_{max}$. For every box, the near-field signatures acquired from non-touching neighbors are always aggregated with the signature inherited from its parent. At the conclusion of the downward pass, every box in $\mathcal{L}_{max}$ will have a near-field signature which represents the fields from all non-touching neighbors and far-away boxes.

Evaluation:

7. For node m, the pressures due to a node n within the same box is evaluated directly and added to the node's self pressure

$$p_{m,box} = i\omega\rho \sum_{n \neq m} q_n \frac{e^{-ik|\vec{r}_m - \vec{r}_n|}}{2\pi|\vec{r}_m - \vec{r}_n|} + \rho c q_m \left[\frac{1}{2\pi}(ka_m)^2 + i\frac{8}{3\pi}ka_m\left(\frac{a_m}{\pi}\right)^{1/2}\right] \tag{24}$$

8. The pressures due to a node n in a neighboring box is also evaluated directly

$$p_{m,neighbors} = i\omega\rho \sum_{n} q_n \frac{e^{-ik|\vec{r}_m - \vec{r}_n|}}{2\pi|\vec{r}_m - \vec{r}_n|} \tag{25}$$

9. The pressures due to the remaining nodes (those in non-touching neighbor and far-away boxes) are included in the near-field signature of the box which is evaluated using (21)

$$p_{m,other} \approx -\rho c \frac{k^2}{8\pi^2} \sum_{\hat{s}} w_s e^{ik(\vec{r}_m - \vec{b})\cdot\hat{s}} N_{\vec{b},L}(\hat{s}) \tag{26}$$

where $\vec{b}$ is the box center, and w_s is the quadrature weight for angle $\hat{s}$.

The total pressure at the node is calculated by summing the contributions from each part.

$$p_m = p_{m,box} + p_{m,neighbors} + p_{m,other}$$

2.3.4 Preconditioning and solving the linear system

Each traversal of the quadtree–consisting of an upward pass, downward pass, and a pressure evaluation–takes as an input the node strengths and returns an approximation of the pressure on each node. To solve the linear system (54), the traversal is paired with a forward iterative solver such as generalized minimal residual (GMRES) or biconjugate gradient (BiCG). These solvers take as an input a guess vector $\bar{w}$ and, with each iteration, attempts to find a better estimate of $\bar{w}$ which minimizes the residual $\vec{p}_{act} - \mathbf{G}\bar{w}$.

Generally, to speed up convergence, the solver is applied to a preconditioned system which has a reduced condition number. Consider the *right* preconditioned system for some preconditioner $\mathbf{P}$

$$\mathbf{G}\mathbf{P}^{-1}\mathbf{P}\vec{w} = \vec{p}_{act} \tag{27}$$

which is solved by first solving

$$\mathbf{G}\mathbf{P}^{-1}\vec{y} = \vec{p}_{act} \tag{28}$$

for $\vec{y}$ and

$$\vec{y} = \mathbf{P}\vec{w} \tag{29}$$

for $\vec{w}$. To be effective, the preconditioned matrix product $\mathbf{G}\mathbf{P}^{-1}$ should have condition number less than that of $\mathbf{G}$. A suitable preconditioner should be: (1) a good approximation of $\mathbf{G}$, and (2) relatively cheap to compute.

Using as much solved information available for our problem, we construct a block-diagonal preconditioner $\mathbf{P}$ where the blocks are the BEM matrices (refer to (54)) for the single membrane problem (wherein no membrane-to-membrane acoustic cross-coupling is assumed).

$$\mathbf{P}^{mn} = \begin{cases} \mathbf{G}^{mn}_{mech} + i\omega \mathbf{Z}^{mn}_{rad} & \text{if m, n are in same membrane} \\ 0 & \text{otherwise} \end{cases}$$

$\mathbf{P}$ and $\mathbf{P}^{-1}$ are sparse and very cheap to construct, with a block size depending on the number of nodes per membrane and a number of matrix inversions equal to the number of unique membrane specifications (typically not more than 2). Note that $\mathbf{P}^{-1}$ is also block-diagonal with blocks that are the inverse of the corresponding blocks in $\mathbf{P}$.

Finally, in solving the system through forward iteration, an error tolerance must be specified to indicate to the solver when to stop. The error is determined based on the relative residual (RR), defined as

$$\text{RR} = \frac{\|\bar{p}_{act} - \vec{p}_{act}\|}{\|\vec{p}_{act}\|} \tag{30}$$

Since our primary concern is with the error in the solved displacement $\bar{w}$, a more useful metric is the relative error (RE), defined as

$$\text{RE} = \frac{\|\bar{w} - \vec{w}\|}{\|\vec{w}\|} \tag{31}$$

The RE is bounded above by the residuals in proportion to the condition numbers $\kappa(\cdot)$ of the preconditioner $\mathbf{P}$ and the right preconditioned system $\mathbf{G}\mathbf{P}^{-1}$.

$$\frac{1}{\kappa(\mathbf{P})} \frac{\|\bar{w} - \vec{w}\|}{\|\vec{w}\|} \leq \frac{\|\mathbf{G}\bar{w} - \mathbf{G}\vec{w}\|}{\|\mathbf{G}\vec{w}\|} \leq \kappa(\mathbf{G}\mathbf{P}^{-1}) \frac{\|\bar{p}_{act} - \vec{p}_{act}\|}{\|\vec{p}_{act}\|} \tag{32}$$

This expression is useful for guessing the tolerance necessary because the condition numbers can be calculated exactly or estimated. $\kappa(\mathbf{P})$ can be found exactly and cheaply from its block-diagonal property, while an estimate for $\kappa(GP^{-1})$ must be assumed.

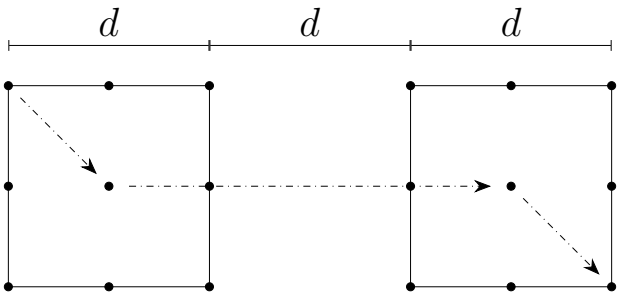


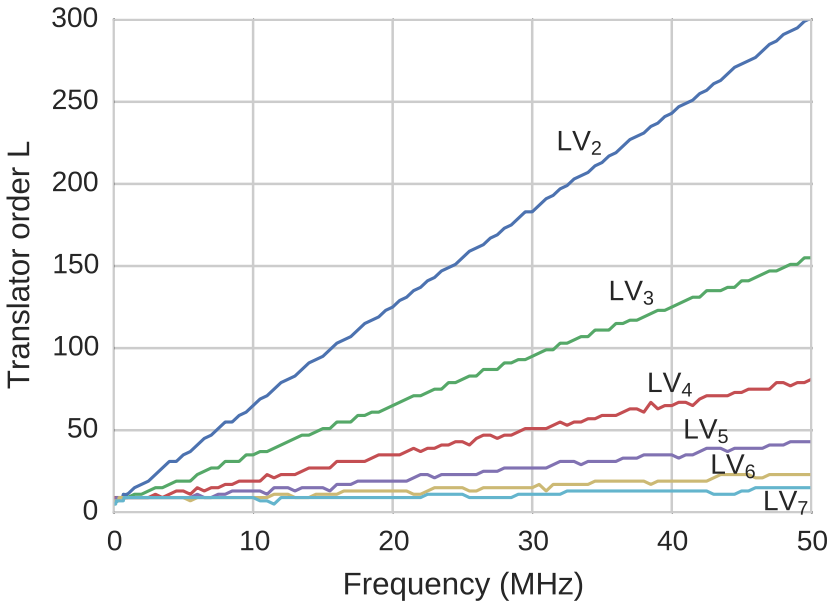
Figure 12: The worst-case translation in the multi-level FMA with a one-box buffer scheme is used to determine the optimal translation order L for each level at each frequency. The worst-case considers the translation from each of the nine locations in the source box to the nine locations in the target box.

2.3.5 Controlling precision

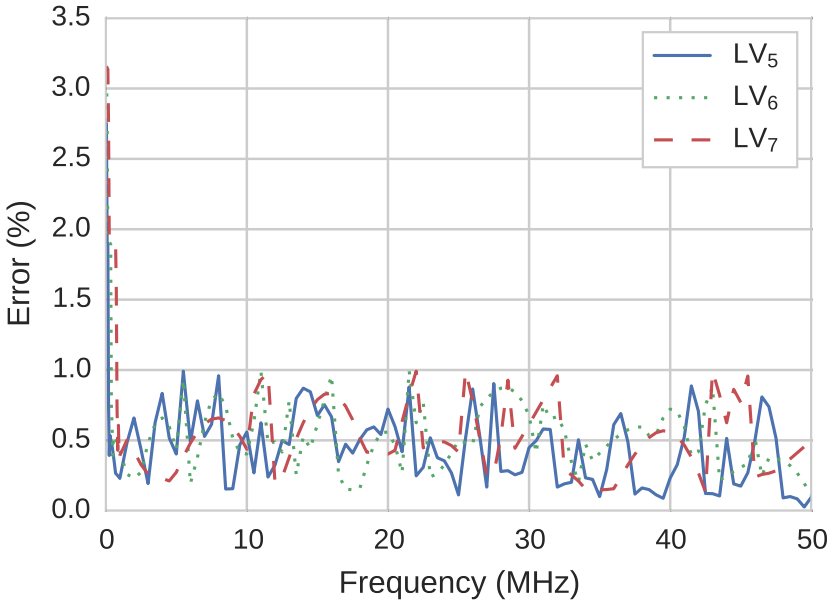
A powerful feature of the FMA is the ability to optimize the trade-off between runtime and accuracy for a particular application. The numerical error incurred by the algorithm come from the truncation of the infinite series and evaluation of the integral by numerical quadrature in (15). The error is controlled primarily by adjusting the order of the truncated operator T_L; enough terms of the series should be kept so that T_L converges within the desired tolerance, but not so many as to incur an unnecessary computational penalty.

A well-known deficiency of the FMA is the susceptibility of the operator T_L to numerical instabilities due to the asymptotic behavior of the spherical Hankel functions [81]. For $l \gg kx$, $h_l(kx)$ diverges, and its representation in floating point introduces round-off error that will compromise the overall accuracy of the algorithm. This effectively imposes an upper bound on the truncation order L, and therefore a limit on the achievable accuracy. The upper bound depends on the product kx_{min} where x_{min} is the minimum distance between box centers which the translation operator will be used. The breakdown therefore becomes a problem at *low* frequencies–where the wavelength is large compared to the size of the boxes–and will also depend on the maximum quadtree level used.

To determine the optimum truncation order at each frequency, we consider the worst-case scenario of a translation from one box to another box separated by a one-box buffer of length d (see Fig. 12). This empirical case also serves to determine the onset of the low-frequency breakdown for a typical problem size and frequency range of interest. Nine sources and nine evaluation locations are simulated (at the center and on the periphery of each box), for a total of 81 combinations. Starting with $L = 3$, the truncation order is increased until either breakdown occurs or the maximum relative error reaches 1% or less. The box sizes and separation distances were selected for a 4×4



(a)



(b)

Figure 13: (a) Optimum translation order L needed to achieve 1% error tolerance as a function of frequency, determined using the worst-case translation. (b) The maximum error in the worst-case translation if the optimum translation order L is used. Breakdown of the translation operator is a problem at frequencies below 780 kHz, limiting the achievable tolerance to about 3%.

mm design space.

The optimal orders determined by this method and the errors incurred are plotted in Fig. 13b and Fig. 13a, respectively. As expected, the error remains within 1% for most frequencies in the range of interest. Breakdown occurs for frequencies below 780 kHz for $\mathcal{L}_7$, 390 kHz for $\mathcal{L}_6$, and 190

kHz for $\mathcal{L}_5$ and the error in the breakdown region does not exceed 3.2%. These small error penalties occur at frequencies that are not relevant for most medical imaging applications.

2.3.6 Numerical quadrature, interpolation, and filtering

Appropriate implementation of numerical quadrature is a critical component of the algorithm and must be treated carefully. Recall that the multipole near-field and far-field signatures are functions of the unit direction vector $\hat{s}$ defined in spherical coordinates by the angles (θ, ϕ) and in Cartesian coordinates by $\langle cos\theta sin\phi, sin\theta sin\phi, cos\phi \rangle$. The integral of the disaggregation step of (21) amounts to an integration of some spherical function $f(\theta, \phi)$ over the unit sphere. That is,

$$\int_0^{2\pi} \int_0^{\pi} f(\theta, \phi)\, sin\phi d\phi d\theta \tag{33}$$

The numerical quadrature scheme which determines the weights and the sampled angles (θ_m, ϕ_n) will ultimately determine the integration error incurred in this step. An optimal scheme should use as few sampled angles as possible while maintaing a sufficiently accurate representation of $f(\theta, \phi)$. Note however that in a multi-level algorithm, because each level is associated with a different cluster radius, an accurate representation of $f(\theta, \phi)$ may necessitate more or fewer sampled angles. For this reason, the quadrature scheme is tied intimately with the procedures of interpolation and filtering.

Since the integration is over spherical harmonics of the form

$$\int_0^{2\pi} \int_0^{\pi} Y_m^l(\theta, \phi) sin\phi d\phi d\theta = \int_0^{2\pi} \int_0^{\pi} e^{im\theta} P_l^m(cos\phi) sin\phi d\phi d\theta \tag{34}$$

a common choice is to choose sampled angles uniformly in θ and at Gauss-Legendre points in ϕ. Doing so will yield a quadrature integration that is exact for spherical harmonics up to some degree [89]. However, the interpolation and filtering procedures [87] for this type of quadrature introduce additional complexities which may be cumbersome for implementation.

A more convenient approach is to utilize trigonometric polynomial (Fourier) basis functions which has the advantages of uniform sample points in both θ and ϕ and that the interpolation and filtering steps can be performed exactly using the Fast Fourier Transform (FFT). To do so, the integration in ϕ is extended so that $f(\theta, \phi)$ is 2π-periodic in both θ and ϕ through the introduction of a spherical property [84]. This results in a $|sin\phi|$ integration weight whose discontinuity in ϕ is smoothed by

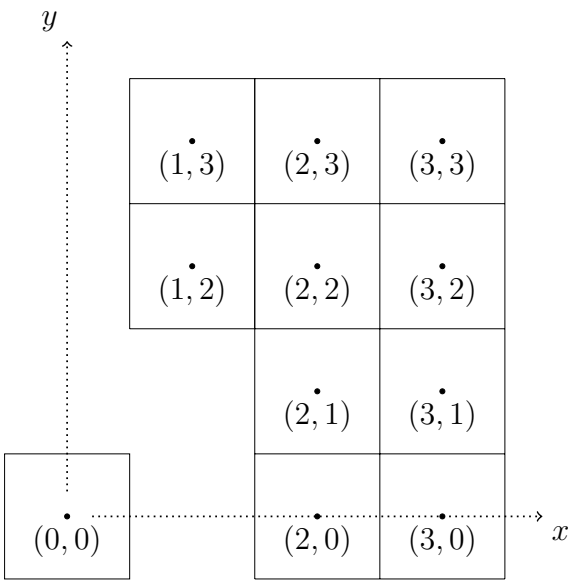


Figure 14: The set of unique translation operators for the 2-dimensional problem.

band-limiting in order to improve convergence [85].

$$\int_0^{2\pi} \int_0^{2\pi} \frac{1}{2} f(\theta, \phi) \, |sin\phi| d\phi d\theta \tag{35}$$

The resulting quadrature rule is given by

$$\theta_m = \frac{\pi m}{M} \text{ for } -M + 1 \leq m \leq M - 1 \tag{36}$$

$$\phi_n = \frac{\pi n}{N} \text{ for } -N \leq n \leq N - 1 \tag{37}$$

where the number of quadrature points M and N are related to the order of the translation operator L by $M = 2L + 1$ and $N = 2(L + 1)$. Storage of the sampled functions can be reduced in half by following the guidelines provided in [85], [84]. The resulting interpolation and filtering procedures are performed simply by zero-padding and truncation, respectively.

2.3.7 Pre-caching of operators

Because an iterative solver will perform a traversal in every iteration, redundant operations should be moved outside the loop to reduce the total computation time (at the expense of a small memory cost). For example, the Green's functions used in the direct pressure evaluation can be computed in advance since they depend only on node-to-node distances which remain static during the iterations.

More importantly, the shift and translation operators used in the upward and downward passes can be computed prior to the iterations as they depend only on the box-to-box geometry. The

precomputation of the translation operators is of particular importance because doing so will isolate the most expensive calculation in the FMA. Because the translations depend only on the relative position between box pairs, the same translation may be encountered multiple times in the algorithm–a redundancy that can be easily avoided. In a multi-level 2-D algorithm with a quadtree structure, there are a total of 40 unique translation operators which may be used (consider the four possible child box positions and the potential non-touching neighbors in each case–there are 40 total translations covering all these cases). If the total number of quadrature angles in azimuth is divisible by 4, symmetry can be exploited to reduce the number of translation operators that need to be computed from 40 to 7 per level (per frequency) with the remaining 33 translation operators constructed from simple rotations or reflections.

2.3.8 Transmit and receive simulation

An integral part of the operation of an imaging array is the simulation of pulse-echo (transmit-receive) events. In the proposed framework, transmit and receive operations are handled by the solution of two separate linear systems.

In transmit, the array nodes are excited by some actuating pressure $\vec{p}_{act}$, determined for CMUTs from (13) or for PMUTs from an approach presented in Chapter 3. The actuating pressure may include both amplitude and phase variations across the array in order to capture apodized and phased operation. By solving the following linear system for node displacements $\vec{w}_{tx}$

$$(-\omega^2\mathbf{M} + i\omega\mathbf{C} + \mathbf{K} + i\omega\mathbf{Z}_{rad})\vec{w}_{tx} = \vec{p}_{act} \tag{38}$$

the fluid-structure behavior of the array is accounted for considering the given excitation topography.

The field pressure from the motion of the array can be calculated by employing the Rayleigh integral for acoustic sources on a baffled plane

$$p(\vec{r}_m, \omega) = -\omega^2\rho \iint_S w_{tx}(\vec{r}_n)\frac{e^{ik|\vec{r}_m - \vec{r}_n|}}{2\pi|\vec{r}_m - \vec{r}_n|}\, dS \tag{39}$$

where ρ is the fluid density, $\vec{r}_m$ is the field location, and $w_{tx}(\vec{r}_n)$ is the normal displacement of the array at location $\vec{r}_n$. Since the displacements are solved for on a nodal surface mesh, the double integral is handled numerically by treating each node as a baffled simple source and summing their

contributions. This yields

$$p(\vec{r}_m, \omega) \approx -\omega^2 \rho \sum_n a_n w_n \frac{e^{ik|\vec{r}_m - \vec{r}_n|}}{2\pi|\vec{r}_m - \vec{r}_n|} \tag{40}$$

where a_n and w_n are the nodal area and normal displacement of the n-th node, respectively. Note that this calculation, carried out in the frequency domain, yields the so-called pressure spatial frequency response (SFR). It is the time-frequency Fourier transform of the spatial impulse response (SIR) [90]. Transmit beamplots (the directivity) of the array can be generated by evaluating (40) over a hemispherical surface.

In receive, the excitation topography of the array is determined from the incident pressure wave $\vec{p}_{inc}$. We may consider generally an incident pressure amplitude and phase which varies arbitrarily over the array. Solving the following system for normal displacements $\vec{w}_{rx}$

$$(-\omega^2 \mathbf{M} + i\omega \mathbf{C} + \mathbf{K} + i\omega \mathbf{Z}_{rad}) \vec{w}_{rx} = \vec{p}_{inc} \tag{41}$$

captures the fluid-structure problem in receive operation.

Strictly speaking, in order to generate beamplots of the array in receive, each field location (point source) produces a unique incident pressure which will necessitate a distinct solution to (41). This procedure quickly becomes cumbersome and prohibitive for detailed beamplots. We consider a reasonable approximation to reduce this computational burden by assuming, for the sake of the fluid-structure problem, that the array is excited by a normal incident wave. The solved displacement amplitude and phase are then modulated as before to account for propagation, apodization and phase from focusing. The Rayleigh integral appears again, but this time the integration is carried out over the receive nodes only, replicating the final summation of the receive signals during beamforming. The resulting mean displacement is given by

$$\bar{w}(\omega) \approx -\omega^2 \rho \sum_{n,rx} a_n w_{n,rx} \frac{e^{ik|\vec{r}_m - \vec{r}_n|}}{2\pi|\vec{r}_m - \vec{r}_n|} \tag{42}$$

2.4 Implementation and validation

We developed an implementation of the fast multipole algorithm for membrane-type arrays in Python [91] with the use of additional open-source packages: SciPy (scientific computing) [92], NumPy (N-dimensional arrays and linear algebra) [93], and Cython (optimising compiler) [94]. The

Loose General Minimum Residual (LGMRES) algorithm [95] implemented in NumPy is used for the iterative solver in our FMA code. Direct solutions are obtained using NumPy's linear algebra solver which is a wrapper for the LAPACK gesv routine [96].

Pre-caching of the translation operators was performed for a 6-level FMA scheme (levels $\mathcal{L}_2$ to $\mathcal{L}_7$) and a 4×4 mm design area. This step took approximately one day of computation on a 12-core computer for a very fine frequency sweep from $0 - 50$ MHz in 0.05 MHz steps. Once computed and stored, the translation operators can be reused for any arbitrary configuration of nodes which fit into the design area. Although the computation of the translation operators takes a significant amount of time, its computational cost is amortized over all subsequent simulations.

Simulations were carried out on a GNU/Linux computer (4x AMD Opteron 6376 CPUs) with 64 effective threads running at 1.4 GHz each and 256 GB of total memory. Frequencies were simulated from $0 - 50$ MHz in 0.25 MHz steps.

2.4.1 Comparison with direct BEM method using a 1-D CMUT array element

A single element of a larger 32-element 1-D CMUT array was simulated to compare our FMA solver with the direct solution. The element consists of a 45×2 grid of CMUT membranes with a pitch of 55 μm in both directions. The CMUT membranes are 45×45 μm squares with a total membrane thickness of 2.2 μm, a silicon nitride isolation layer of 0.2 μm, gap of 47 nm, and a damping coefficient of 10,000 $Pa \cdot s/m$. Each membrane was meshed with a 13×13 grid of nodes (excluding clamped nodes), chosen such that the total node count of 15,210 is just within the limit of feasibility for the direct solver. At each frequency, the membranes were given a 9 V DC bias and were excited uniformly with a 1 V AC signal (a plane wave excitation). The simulations were performed under fluid loading conditions with a fluid density of 1000 kg/m^3 and a sound speed of 1540 m/s. These membrane and fluid properties were used in all subsequent simulations unless otherwise noted.

The nodal displacements were calculated using our FMA solver and compared with the direct solution. Two measures were used to determine the displacement error: normalized root mean squared error (NRMSE) which normalizes to the range of observed values and the relative error

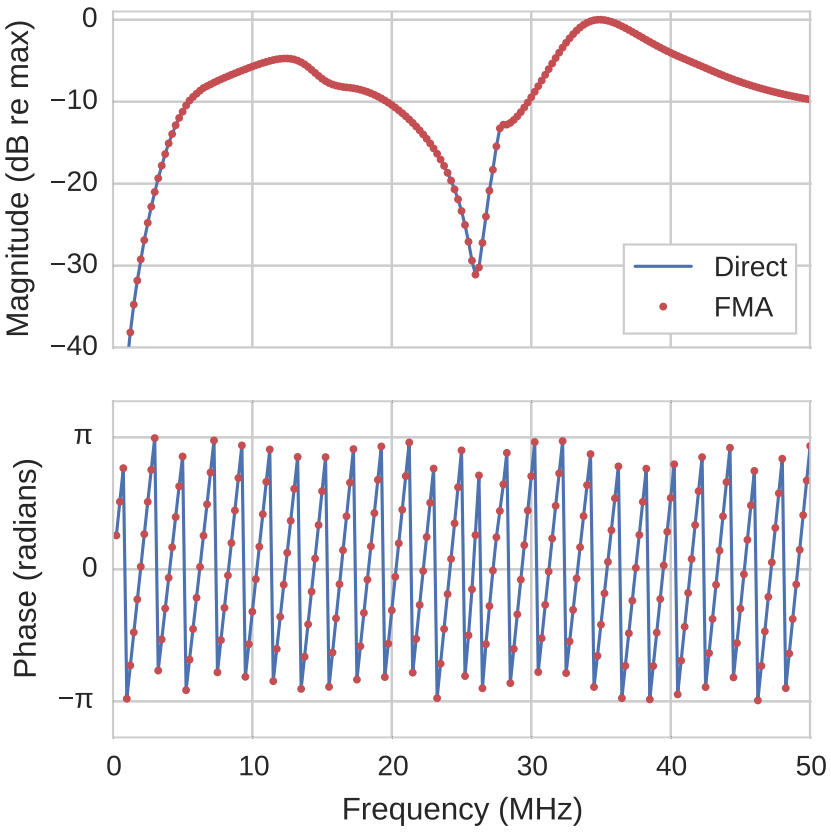


Figure 15: Pressure response magnitude (top) and phase (bottom) for the simulated element at 3 cm from the center of the array. The membranes of the element were given a 9 V DC bias and excited uniformly.

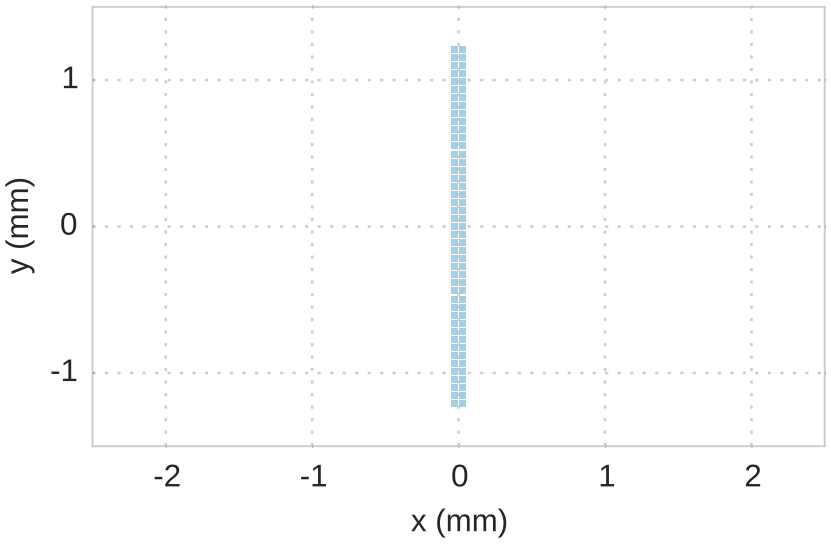


Figure 16: A single element of a 1-D CMUT array with 90 membranes (arranged in two columns) and 15,210 nodes. The membranes are 45 $\times$ 45 μm squares with a pitch of 55 μm. Because the BEM equations for a simulation of this size can be solved directly, this array is used to validate our FMA solver.

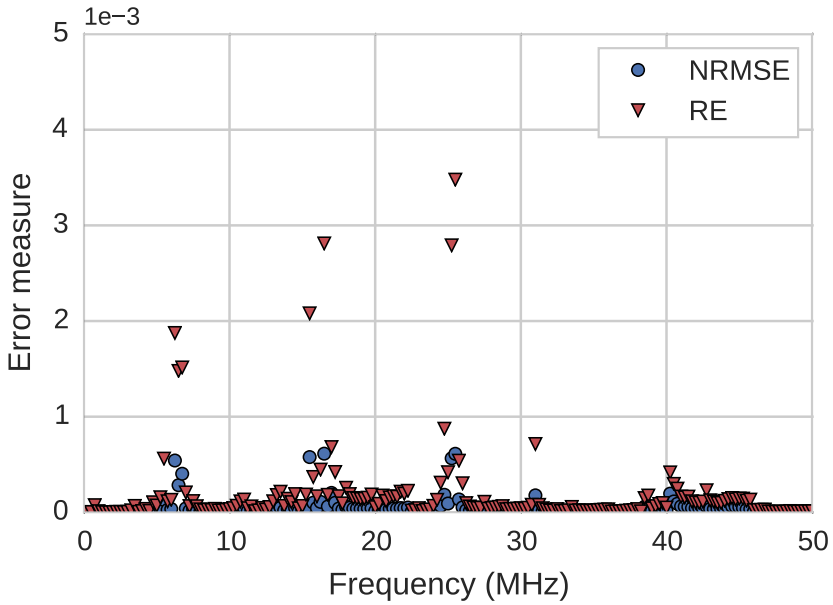


Figure 17: Normalized root mean squared error (NRMSE) and relative error (RE) for the node displacements of the simulated element. The element was simulated with FMA and compared with the direct solution. The agreement is within 0.5% at all frequencies.

(RE) which normalizes to the *l*-2 norm.

$$\text{NRMSE} = \frac{\sqrt{\frac{1}{N}\sum(|\hat{x}_{FMA}| - |\hat{x}_{direct}|)^2}}{max(|\hat{x}_{direct}|) - min(|\hat{x}_{direct}|)}$$

$$\text{RE} = \frac{\|\hat{x}_{FMA} - \hat{x}_{direct}\|}{\|\hat{x}_{direct}\|}$$

The NRMSE and RE at each frequency is plotted in Fig. 17. We can see that the FMA solution has converged sufficiently to the direct solution with errors much less than 1%. With the block-diagonal preconditioner, the FMA solver converges rapidly with an average of 2.5 iterations of LGMRES and a maximum of 4 iterations. Several peaks in the error are observed at frequencies which likely correspond to particular resonances (and anti-resonances) of the array or the membrane where the conditioning of the system is highest.

The pressure response at 3 cm from the center of the array is shown in Fig. 15. The FMA solver recovers both pressure magnitude and phase with close agreement.

2.4.2 Computation time and memory usage

Resource usage was measured at each frequency for both our FMA solver and the direct solver. To ensure fairness and accuracy, each frequency was simulated in its own process which was limited to running in a single thread. Solution time for the FMA solver includes the necessary FMA-related

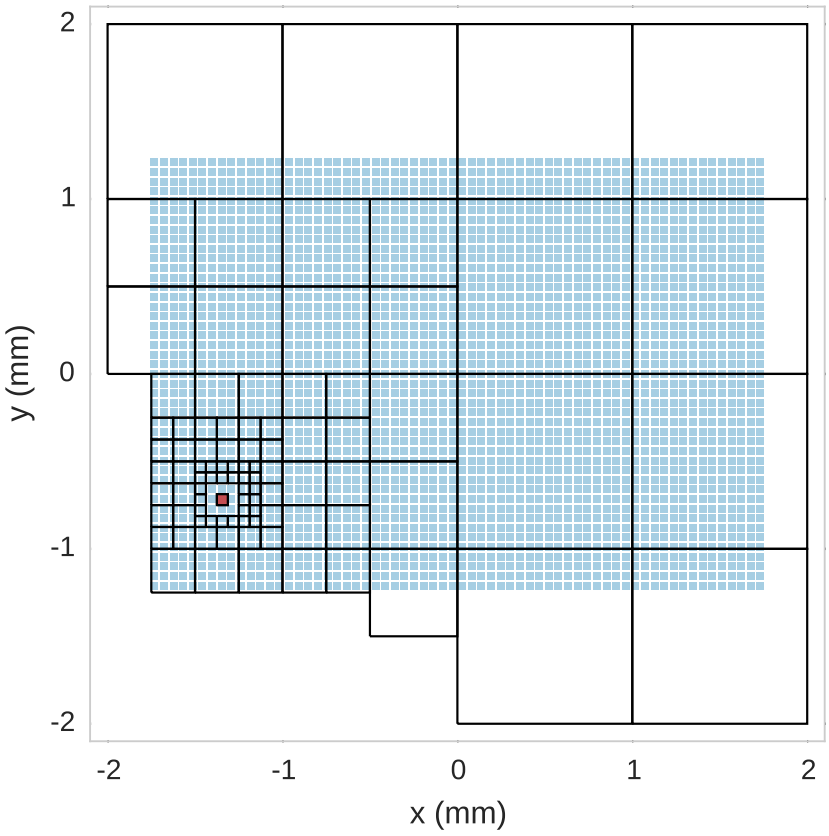


Figure 18: An example of the box-to-box interactions of the multi-level FMA used in the simulation of a 1-D CMUT array. The nodes of the array are assigned to boxes (shown in a black outline) based on the subdivision of a 4×4 mm space using a quadtree. The source boxes get larger as they get farther away from the target box (shown in red) in order to reduce the total number of interactions that need to be calculated.

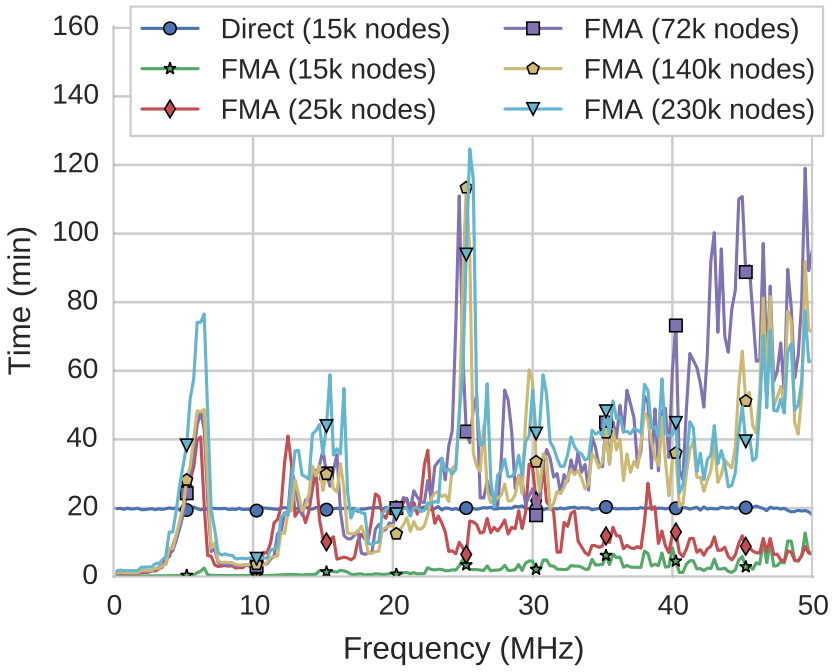
overhead, e.g. the setup of the quadtree, loading of the translation operator pre-cache, and the time spent in iterations of LGMRES. The time spent constructing the translation operator pre-cache is not included. Since the design space is generally known in advance and because the pre-cache is reusable for all simulations fitting into the same space, this step is considered as a one-time cost. Solution time for the direct solver includes the generation of the mutual impedance matrix and the time spent performing LU factorization. In all cases, memory usage is reported as the peak usage by the process during its lifetime.

The solution time for all the simulations are shown in Fig. 19a. For the simulation of a single element with 15k nodes, the solution time of the direct solver was constant at around 21 min per frequency. In comparison, the FMA solver averaged about 2 min per frequency with a maximum of 13 min, a 10-fold reduction in the average computation time. For a full 32-element 1-D CMUT array with 230k nodes, the FMA solver spent on average 33 min per frequency and a maximum of 125 min.

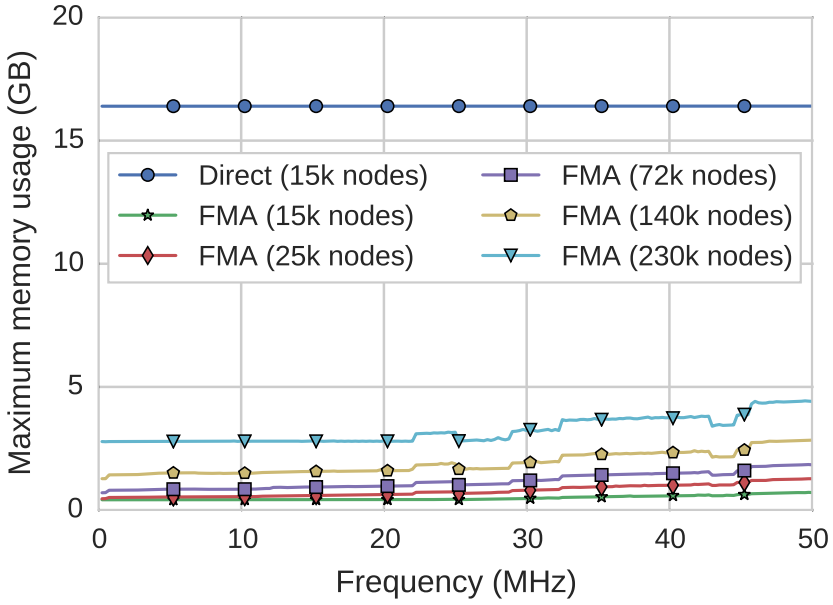
Similarly, the FMA solver shows a significant improvement over the direct solver in memory usage (see Fig. 19b). For 15k nodes, the direct solver uses an average of about 16.4 GB per frequency whereas the FMA solver uses an average of only 500 MB, a 32-fold improvement. The memory usage increases for the larger cases but never exceeds 5.0 GB per frequency. The memory usage is observed to increase as a function of frequency as a result of the finer quadrature sampling needed.

2.5 Summary

We applied the fast multipole algorithm to improve the computational efficiency of a BEM model for membrane-type ultrasonic transducers. With an FMA-accelerated BEM model, simulations of large arrays with thousands of membranes is realized without constraints on the finiteness, periodicity, membrane-type, or phasing of the array. Crucially, by including a mutual radiation impedance term, the model captures the acoustic cross-talk of arrays which produce the dispersive-guided modes that may degrade imaging performance. For a single array element with 90 membranes and 15,210 nodes, our FMA solver was found to be 10 times faster and 32 times more memory efficient than a standard solver using LU decomposition. This improvement was demonstrated over a wide frequency range up to 50 MHz. We demonstrated the ability of our FMA solver to handle large arrays by simulating a full 32-element CMUT array with 2880 membranes and 233,280 nodes.



(a)



(b)

Figure 19: (a) Comparison of simulation times for the FMA solver and direct solver. The FMA solver is about 10 times faster for the simulation of a single element and remains within reasonable speeds for simulations of the full array. (b) Comparison of peak memory usage for the FMA solver and direct solver. The FMA solver uses around 32 times less memory for the simulation of a single element. The memory usage for the full array remains below 5.0 GB and within the range of feasibility.

CHAPTER 3

A HYBRID BOUNDARY ELEMENT MODEL FOR PMUT ARRAYS

3.1 Background and motivation

In recent years, considerable focus has been placed on the design and manufacture of PMUT arrays. Unlike their capacitive counterpart, PMUTs do not require bias voltages, have a higher capacitance and lower electrical impedance suitable for integration with low voltage electronics, and behave linearly within typical operating parameters. Looking at just the past few years, PMUT arrays have been demonstrated in devices for applications such as wireless energy transfer [97], ultrasonic fingerprint sensing [31], and intracardiac imaging [98]. With expanding interest in large PMUT arrays, there exists a need for practical and accurate simulation tools to aid in the design process.

To date, many authors have contributed to the growing literature on PMUT modeling and optimization. Perhaps the most fundamental approach is to consider analytical solutions based on boundary-fixed thin circular disks. For instance, in [99], differential equations were derived to describe the static deformation of a piezoelectric unimorph actuator with full electrode coverage. Solutions were obtained by considering axisymmetric deflection of the PMUT with simple support boundary conditions under a uniform load or concentrated load. In [100], general solutions in the form of Bessel (and modified Bessel) functions were obtained for the case of harmonic excitation of a unimorph PMUT with partial piezoelectric and electrode layers. The addition of partial layers introduces extra boundary conditions which must be satisfied in conjunction with the condition for the perimeter. Analytical solutions for other PMUT geometries have also been explored, including curved PMUTs with spherical diaphragms [101] and bimorph PMUTs actuated with opposite phase [102].

Another effective approach, particularly suitable for geometries with multiple electrodes, is to consider normal mode solutions to the governing equation of motion. For example, in [103], a lumped electromechanical model was obtained by consideration of the first mode shape only of a rectangular clamped membrane. In [104,105], the problem of multiple layer, multi-electrode PMUTs was explored, wherein a Green's function solution was proposed and solved by projection onto the normal modes of a clamped circular disk. A similar approach was used by Sammoura et. al. to develop an equivalent circuit model, with the addition of residual stress and consideration only of the first two axisymmetric modes [106]. Finally, a numerical approach was proposed based on a build-test finite element model, where parameters such as coupling coefficient, acoustic impedance,

and resonance frequency are extracted from the model for the purpose of optimization [107].

In all the preceding contributions, the focus has been on accurate modeling of single PMUTs in vacuum and air, or in fluid by first-order approximation of the PMUT as a circular piston radiator. However, when placed in the context of an array, it is well known that membrane-based transducers in immersion will be affected by fluid-born and potentially substrate-born mechanical waves which couples the dynamics of the individual membranes together. In CMUT arrays, the phenomena of acoustic cross-talk has been characterized extensively in simulation and in experiment [54–56]. It has been found that these coupled waves can drastically change the dynamic response of CMUTs and create interference in the primary band that degrades the bandwidth and directionality performance of the transducer. Accurate simulation of these physics is therefore critical to the design and optimization of membrane-type transducer arrays.

Unfortunately, for large arrays composed of hundreds of individual membranes, simulation with finite element method (FEM) is computationally prohibitive. In some restrictive cases, assumptions can be made about the periodicity and symmetry of the problem to reduce the total mesh size [55,57,58]. Other authors have developed electromechanical mesh network models where membrane-to-membrane interactions are calculated from analytical expressions for fixed-boundary circular disks [60–62]. Specifically, the mutual radiation impedance of two fixed-boundary vibrating disks is derived by assuming axisymmetric deflection profiles represented by finite power series which satisfy the boundary condition at the disk perimeter. This modeling approach is also employed by Akhbari et. al. in a notable recent work on PMUT arrays [63].

In this chapter, we develop a practical simulation tool for large PMUT arrays with a focus on the accurate capture of acoustic cross-talk. A boundary element method (BEM) simulation is employed where membrane motion is represented by a surface mesh. Membrane dynamics such as stiffness and piezoelectric load are extracted from finite element method (FEM) simulation of a single membrane structure. Acoustic interactions are calculated using a multi-level fast multipole algorithm (ML-FMA), which we have reported on previously [108]. This approach has a number of distinct advantages when compared with previous models. First, because acoustic interactions are calculated on a node-to-node basis, membranes are not restricted to simple axisymmetric deflection. In fact, we have shown previously that high-order modes and anti-symmetric modes may contribute to cross-talk, even within the first frequency band of the transducer [108]. Second, by utilizing FEM simulation, the BEM model is generalized to cover more complicated membrane structures. Here, the only restrictions on the membrane structure is that it is approximately flat, it operates

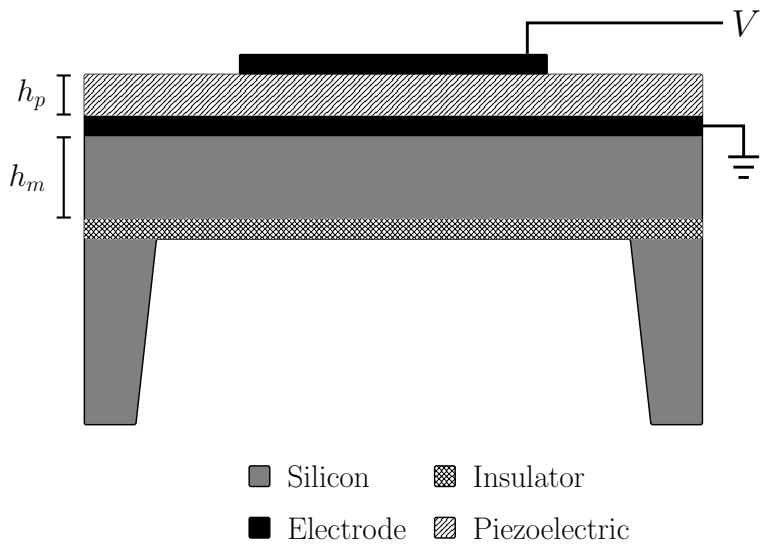


Figure 20: Cross-section of a standard PMUT design.

linearly about its static position, and that it can be simulated feasibly with FEM. Although we do not demonstrate all these cases here, the method applies equally to PMUT geometries featuring small curvature (domed), pre-stressed layers, mass-loading, multiple layers, low aspect ratio (thick), and complex boundary terminations. Finally, through the application of a multi-level fast algorithm, arrays with thousands of membranes can be simulated with reasonable computational resources.

The structure of this chapter is as follows. The methods of the proposed hybrid boundary element model are described, including a review of the thin-plate model for PMUTs and a description of the methods for extracting model parameters from FEM. The model is validated by comparison with FEM simulation for a 3 by 3 array for both circular and square membrane geometries. Finally, discussion and concluding remarks are given.

3.2 Methodology

3.3 Analytical thin-plate model for PMUTs

Micromachined piezoelectric transducers based on a membrane-type design (sometimes referred to as flextensional) are composed of alternating layers of active (in the piezoelectric sense) and passive materials suspended over a vacuum or air-filled gap. When the active layers are driven by an electric field, uneven expansion and contraction between the layers results in deflection of the membrane. A standard PMUT design is depicted in Fig. 20. We consider here, without loss of generality, a simplifying case of this design: a single elastic layer of thickness h_m bonded to a piezoelectric layer of thickness h_p.

Classical thin-plate theory to PMUTs has been applied successfully to described PMUT behavior [99, 100], This analytical model is briefly reviewed here. In this limit, the effect of an applied electric field to the piezoelectric composite membrane is to induce a bending moment, and therefore a bending deflection. For rectangular PMUT geometries, the moment resultants are expressed in terms of the plate displacement w and applied electric potential $V(x, y)$ as

$$M_x = -D\left(\frac{\partial^2 w}{\partial x^2} + \nu\frac{\partial^2 w}{\partial y^2}\right) - \frac{d_{31}Y_p(z_2 + z_1)}{2(1-\nu_p)}V(x, y) \tag{43}$$

$$M_y = -D\left(\nu\frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial x^2}\right) - \frac{d_{31}Y_p(z_2 + z_1)}{2(1-\nu_p)}V(x, y) \tag{44}$$

$$M_{xy} = -D(1-\nu)\frac{\partial^2 w}{\partial x \partial y} \tag{45}$$

where D the is flexural rigidity of the composite plate, ν is its mean Poisson's ratio, and Y_p, ν_p, and d_{31} are the Young's modulus, Poisson's ratio, and piezoelectric strain coefficient of the piezoelectric material, respectively. The coordinates $z_1 = h_m - c$, $z_2 = h_m + h_p - c$ are related to the coordinate of the neutral plane c. For a rectangular top electrode bounded by $x = \pm x_e$ and $y = \pm y_e$, the applied electric potential function with amplitude V_0 is prescribed in terms of step functions

$$V(x, y) = V_0(H(x + x_e) - H(x - x_e)) \times (H(y + y_e) - H(y - ye)) \tag{46}$$

The equation of motion is

$$\begin{aligned}\frac{\partial^2 M_x}{\partial x^2} + 2\frac{\partial^2 M_{xy}}{\partial x \partial y} + \frac{\partial^2 M_y}{\partial y^2} + (\rho_m h_m + \rho_p h_p)\frac{\partial^2 w}{\partial t^2} \\ = -D \cdot \nabla^4 w + (\rho_m h_m + \rho_p h_p)\frac{\partial^2 w}{\partial t^2} \\ = p_{piezo}\end{aligned} \tag{47}$$

where ∇^4 is the biharmonic operator, ρ_m is the density of the elastic material, ρ_p is the density of the piezoelectric material, and p_{piezo} is the piezoelectrically-induced loading of the plate.

Substituting (43) – (46) into (47), p_{piezo} assumes the following form

$$\begin{aligned}p_{piezo} = \frac{d_{31}Y_p V_o(z_2 + z_1)}{2(1-\nu_p)}\Big[\big(\delta'(x + x_e) - \delta'(x - x_e)\big) \times (H(y + y_e) - H(y - y_e)) + \\ \big(\delta'(y + y_e) - \delta'(y - y_e)\big) \times (H(x + x_e) - H(x - x_e))\Big]\end{aligned} \tag{48}$$

The normal load representation can be understood as the application of a pair of opposing

normal loads at the boundaries of the top electrode, creating a bending moment with magnitude in proportion to the applied voltage. The amplitude of the load is proportional to the layer thicknesses, the mechanical properties of the piezoelectric material and its piezoelectric constant, and the input voltage.

Upon inspection of (48), the appearance of step functions and delta function derivatives makes numerical implementation difficult, if not impossible. A natural idea is to smooth the discontinuities of this function by filtering or perhaps by employing sigmoid functions. However, to achieve reasonable accuracy, such an approach would necessitate very fine spatial sampling around sharp features that would greatly increase mesh size to the point of impracticality for large simulation. Rather than attempt to find a suitable analytical approximation of (48), we seek a numerical equivalent obtained by utilizing FEM simulation of a single membrane structure. This procedure is detailed in the following section.

3.3.1 Boundary element method simulation for membrane-type transducers

A numerical model of membrane motion based on the boundary element method was described in Chapter 2. As a review, the BEM model involves the solution of the following linear system of angular frequency ω, vertical displacement $\vec{w}$, and actuating load $\vec{p}_{act}$

$$(-\omega^2\mathbf{M} + i\omega\mathbf{C} + \mathbf{K} + i\omega\mathbf{Z}_{rad})\vec{w} = \vec{p}_{act} \tag{49}$$

Here, $\mathbf{M}$ is a mass matrix with diagonal entries representing the mass per unit area of each node, i.e. $M_{ii} = \rho_m h_m + \rho_p h_p$ in the case of a composite plate, where ρ_m is the density of the elastic material and ρ_p is the density of the piezoelectric material. $\mathbf{C}$ is a damping matrix with diagonal entries $C_{ii} = \beta$ for some damping coefficient β. $\mathbf{K}$ is the stiffness matrix representing the coupled-stiffness of the membrane nodes and $\vec{p}_{act}$ is the actuating load which excites the membrane into vibration. In the case of PMUTs, $\vec{p}_{act} = \vec{p}_{piezo}$ is a result of the application of an electric potential. For thin membranes, this effect can be expressed analytically by (48).

In a hybrid BEM model, some of these matrices are extracted from FEM simulation as opposed to derivation from analytical equations. It will be shown that, in this manner, $\mathbf{K}$ can be derived more accurately and for a wider range of membrane geometries and that a numerical representation of $\vec{p}_{act}$ can be determined.

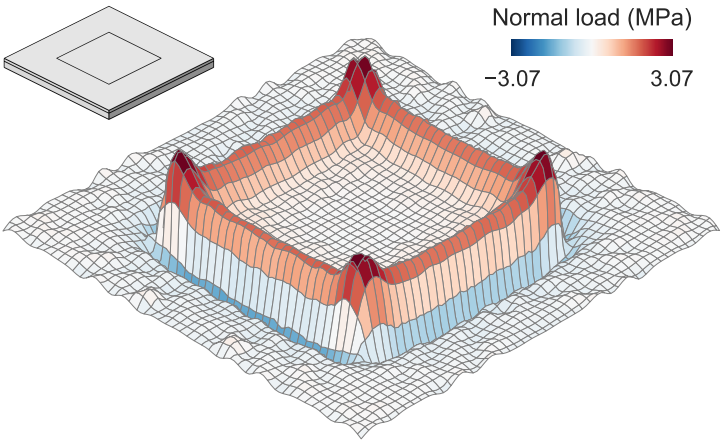


Figure 21: Numerical representation of the piezoelectrically-induced load $\tilde{p}_{piezo}$ for a square membrane.

3.3.2 Extracting parameters from finite element method simulation of a single membrane

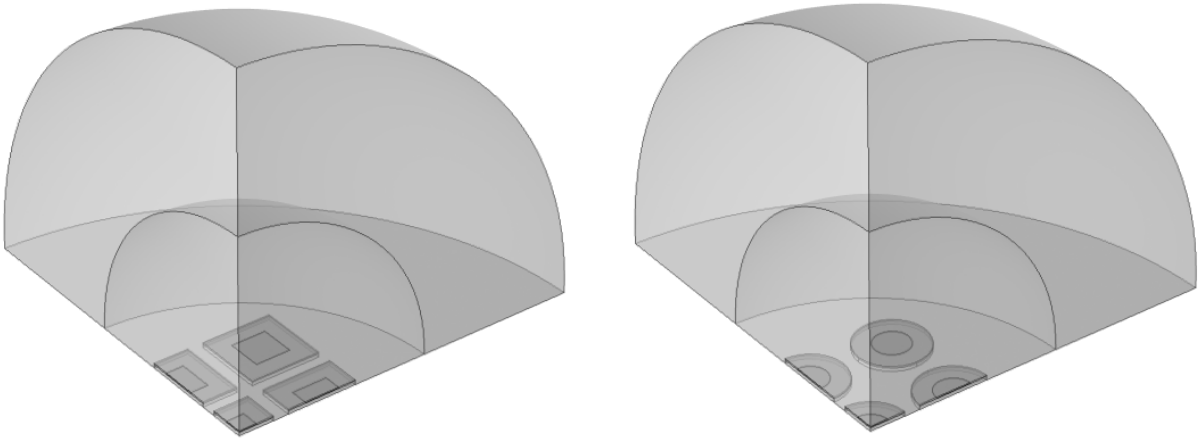
The piezoelectrically-induced loading $\vec{p}_{piezo}$ is determined by matching BEM simulation and FEM simulation of a single membrane structure. Consider the linear system for a single membrane with mass, stiffness and radiation impedance denoted by $\mathbf{M_1}$, $\mathbf{K_1}$, and $\mathbf{Z_1}_{rad}$, respectively.

$$(-\omega^2\mathbf{M_1} + \mathbf{K_1} + i\omega\mathbf{Z_1}_{rad})\vec{w} = \vec{p}_{piezo} \tag{50}$$

We build, as closely as possible, a multi-physics model of the membrane in FEM software and solve for the case of 1 V harmonic excitation. Note that we have chosen to omit mechanical damping in both BEM and FEM, although damping from radiation has been retained in order to avoid sharp resonance peaks. The displacement of the membrane $\vec{w}_{fem}$ is extracted from FEM at grid points corresponding to the BEM surface mesh. With this solution in hand, (50) is solved for $\vec{p}_{piezo}$, obtaining a numerical approximation which can be considered exact for the given boundary element model and surface mesh.

$$\vec{p}_{piezo} \approx (-\omega^2\mathbf{M_1} + \mathbf{K_1} + i\omega\mathbf{Z_1}_{rad})\vec{w}_{fem} \tag{51}$$

In this way, at each frequency, it is ensured that the displacement $\vec{w}_{fem}$ for a single membrane is enforced in the boundary element model. This simple but effective procedure results in an efficient framework which combines the general power of FEM with the numerical efficiency of BEM.



(a) Square membranes (b) Circular membranes

Figure 22: COMSOL models of a 3 by 3 PMUT matrix array with square and circular membranes.

Here, FEM handles the complex electromechanical coupling which determines the single membrane behavior, while BEM handles the large-scale fluid-structure radiation which determines the overall behavior of the array. An example of the piezoelectrically-induced load obtained using this method is shown in Fig. 21. Naturally, the accuracy of this approach will depend on the ability to correctly match the relevant physics in both BEM and FEM.

FEM simulation can also be used to determine the stiffness matrix $\mathbf{K}$ for membrane geometries that are not described accurately by thin-plate theory. Note that $\mathbf{K}$ is block-diagonal with blocks $\mathbf{K_1}$. The following procedure was first proposed by Zahorian et. al. [27]. Static deformation of the membrane is calculated under a 1 Pa load applied individually to each node of the membrane. Care is taken to ensure that the boundary element and finite element nodes correspond to the same locations. For each applied load, the displacements calculated on the node grid of N total nodes are used to form the columns of the compliance matrix $\mathbf{K}^{-1}$. That is,

$$\mathbf{K_1} = \begin{bmatrix} & & & \\ \vec{w}_1 & \vec{w}_2 & \cdots & \vec{w}_N \\ & & & \end{bmatrix}^{-1} \tag{52}$$

where $\vec{w}_n$ is the displacement for loading of the n-th node. By obtaining stiffness $\mathbf{K_1}$ and piezoelectrically-induced load $\vec{p}_{piezo}$ from FEM, simulation of common PMUT designs, such as those with stepped layers (e.g. mass-loaded), low aspect ratios, and multiple layers, is possible.

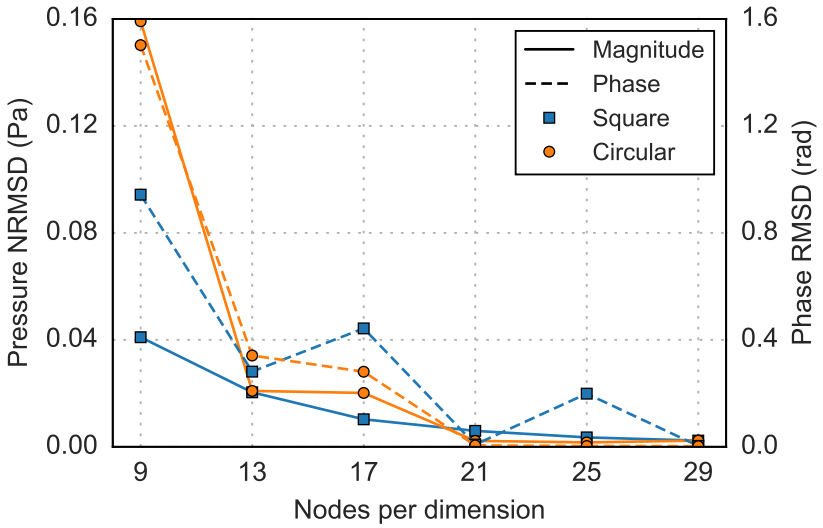


Figure 23: Normalized root mean square delta (pressure) and root mean square delta (phase) showing convergence behavior of hybrid BEM as a function of increasing node density. The pressure was calculated at a distance of 1 mm for a 3 by 3 array.

3.4 Validation with finite element method software

For validation purposes, a 3 by 3 matrix array was simulated using both the proposed hybrid method and finite element software (COMSOL multiphysics, COMSOL Group, Burlington, MA).

The simulated geometries are shown in Fig. 22. The simulated membranes consisted of a 1 μm thick lead zirconate titanate (PZT) layer deposited on a 2.2 μm silicon oxide layer, both extending across the entire membrane. For simplicity, the default material properties from COMSOL were used (specifically, bulk PZT-5H and isotropic SiO_2 with $\rho_m = 2200\ kg/m^3$, $Y_m = 80\ GPa$ and $\nu_m = 0.17$). To avoid the meshing of very thin layers in FEM, the electrode layers (typically on the order of $\sim$100 nm) were omitted from this analysis. The array pitch was 55 μm in both x and y dimensions. Both square membranes (45 by 45 μm) and circular membranes (45 μm diameter) were simulated, with 50% top electrode coverage with respect to the lateral dimension. All membranes were excited in phase with a 1 V harmonic potential for frequencies over a wide range from 0 to 50 MHz. Due to the extensive computation time required, a coarse 500 kHz step was used in COMSOL simulation, with extra refinement (50 KHz step) around significant features. A fine 50 kHz step was used for all simulations with hybrid BEM.

First, mesh convergence behavior of hybrid BEM was investigated by varying the number of nodes from 5 to 29 nodes per membrane dimension (see Fig. 23) Pressure was calculated from displacement via the Rayleigh integral at a distance of 1 mm above the array. Convergence behavior

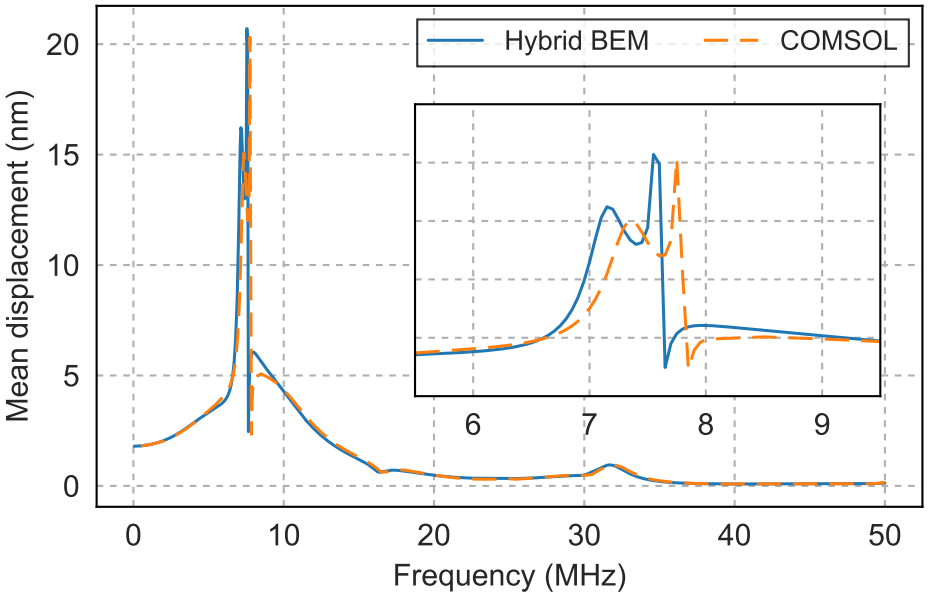


Figure 24: Mean displacement of the center membrane for a 3 by 3 matrix array simulated with hybrid BEM and with COMSOL.

was quantified by observing the change in the solution as the node density is increased. Specifically, a normalized root mean squared difference (NRMSD) pressure magnitude and a root mean squared difference (RMSD) phase are defined as

$$\text{NRMSD} = \frac{\sqrt{\sum_{w,f} \left(|p_n| - |p_{n-1}|\right)^2}}{|p_n|_{max}} \tag{53}$$

$$\text{RMSD} = \sqrt{\sum_{w,f} (\phi_n - \phi_{n-1})^2} \tag{54}$$

where $|p|$ is the pressure magnitude, ϕ is the pressure phase, the subscripts n and $n-1$ denote the current and previous node densities, respectively, and the sum is taken over all nodes and all frequencies. The pressure magnitude is normalized to the maximum observed value $|p_n|_{max}$ to counter the effect of geometric spreading. At a mesh density of 13 nodes per membrane dimension, the magnitude change is observed to be less than 2% and the phase change less than 0.4 radians.

Next, the simulated displacements from hybrid BEM (square membranes) are compared with the reference results from COMSOL simulation. The mean displacement of the center membrane is plotted as a function of frequency for both cases (see Fig. 24). The comparison indicates that hybrid BEM accurately captures the displacement behavior of the array, predicting the resonance features in the 6 - 9 MHz band, and also around 32 MHz. However, small discrepancies (less than 300 kHz) are observed in the predicted frequency at which these features occur. These discrepancies

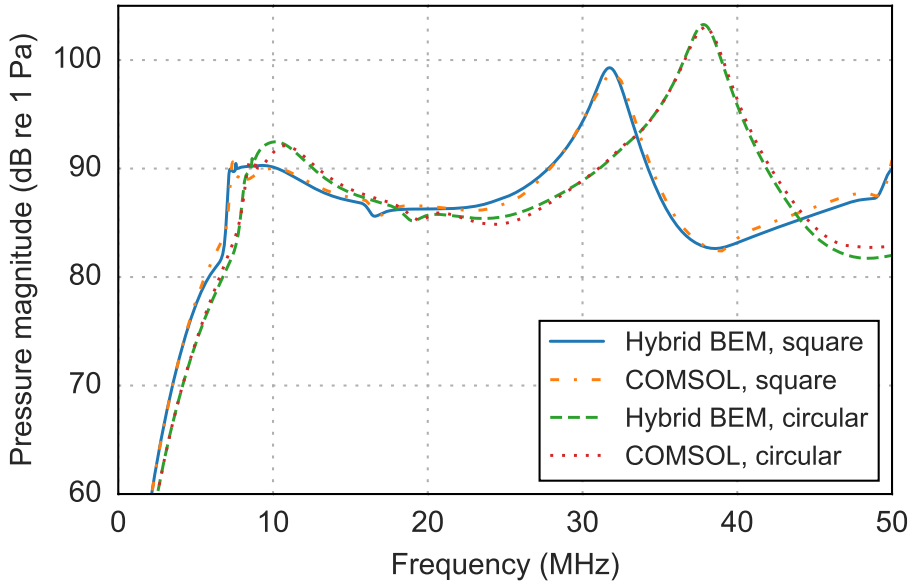


Figure 25: Comparison between the proposed hybrid BEM and COMSOL simulated pressure response for a 3 by 3 array.

are attributed to approximation error resulting from the point source model employed in BEM.

Finally, the pressure response calculated at a distance of 1 mm from the center of the array is shown in Fig. 25. For both square and circular membranes, the pressure response shows good agreement between hybrid BEM and COMSOL simulation. Here, the small discrepancies in the displacement response are mostly reduced by the integration over the surface of the array.

3.5 Summary

We have introduced a hybrid boundary element model for the simulation of large PMUT arrays. Model parameters of membrane stiffness and piezoelectrically-induced load are extracted from finite element method simulation. Simulation of thousands of membranes is realized through the use of a multi-level fast multipole algorithm. The model was validated against finite element method simulation (COMSOL) for the case of a 3 by 3 matrix array with either square or circular membrane geometries. Through convergence analysis, it was determined that a node density of 13 nodes per membrane dimension was sufficient for a pressure NRMSD below 4% and phase RMSD below 0.4 radians for a wide frequency range of 0 - 50 MHz. Surface displacement indicated good agreement between hybrid BEM and FEM.

CHAPTER 4

EXAMPLES OF LARGE ARRAY SIMULATION AND OPTIMIZATION

The computational capabilities of the FMA-accelerated BEM model described in Chapter 2 and the hybrid BEM method described in Chapter 3 enable a variety of large array simulations that are of academic interest. Common geometries for realistic imaging arrays, such as 1-D linear and 2-D matrix arrays, can be simulated in their entirety. In this chapter, simulation, optimization, and analysis of cross-talk behavior are demonstrated for select CMUT and PMUT array designs. The following examples were simulated: a 32-element CMUT linear array to study mesh convergence behavior, surface topography, channel cross-talk, element directivity, and bandwidth broadening; a 7 by 7 PMUT matrix array to study the effects of pitch on output pressure and radiation resistance; and a 32-element PMUT linear array to investigate material choice and top electrode coverage optimizations.

4.1 32-element CMUT linear array

A large 32-element 1-D CMUT linear array, representative of a standard array design for 2-D imaging, was simulated with the properties summarized in Table 3. The array geometry is shown in Fig. 26. Each element was composed of a 2 by 45 grid of CMUT membranes with a pitch of 55 μm in both directions. The elements were arranged linearly with element pitch of 110 μm which satisfies the $\lambda/2$ criterion for a 7 MHz imaging array. Total membrane count was 2880, simulated using up to 233,280 nodes.

4.1.1 CMUT mesh convergence analysis

To establish mesh convergence and investigate the effect of higher-order membrane modes (see Fig. 27), simulations were performed for membrane mesh grids of 3×3, 5×5, 7×7, and 9×9 moving nodes. The total node count for each simulation was 25,920, 72,000, 141,120, and 233,280 nodes, respectively. It is predicted that a finer node density will improve the sampling of higher-order membrane modes and array edges which should increase the accuracy of the simulation at high frequencies.

First, the array was simulated at each node density with a uniform excitation of the first element only (the leftmost element). The single element excitation case is useful for understanding the cross-

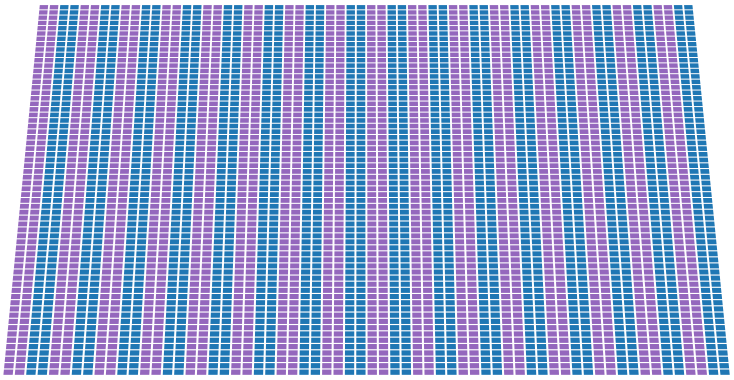


Figure 26: A 32-element CMUT linear array.

talk tendencies of the array and also for practical scenarios such as synthetic beamforming where a small number of elements are excited sequentially. The excited element was given a 9 V DC bias, while the remaining elements were unbiased.

For this case, the pressure response was calculated at 3 cm from the center of the array and is shown in Fig. 28, where w is the membrane width. Significant cross-talk is observed in the 3 - 7 MHz range (the single membrane fundamental mode is around 6.3 MHz), related to modes of the array similar to those predicted by eigenanalysis of small CMUT arrays [56]. Interestingly, while the crosstalk is predicted in all the simulations, convergence is not observed until a sampling length of $w/8$ or smaller is used. 2-D images of the mean membrane displacement magnitude at 5.5 MHz (see Fig. 29) reveal critical differences in the structure of the predicted array mode between the $w/4$ and $w/10$ meshes, likely due in part to insufficient sampling of the array edges. Many membranes were also found to be moving in higher-order modes, which would not be properly sampled by the coarse mesh. The displacement profiles, constructed from the displacement of the center node of each membrane, along the leftmost column and bottommost row is shown in Fig. 30. The results indicate that a fine mesh density, about 7 nodes over the width of the membrane, is necessary to accurately predict crosstalk effects, even in the lower operating band of the transducer.

Strong crosstalk is also predicted in the 14 - 17 MHz range (see Fig. 28), related to cross-coupling of higher-order membrane modes. The difference in the predicted frequency of the crosstalk differs by nearly 2 MHz between the $w/4$ and $w/10$ meshes, indicating the significance of the mesh density on the accuracy of the result. A detailed plot of the membrane motion is shown in Fig. 31 for a frequency of 16.5 MHz, confirming the presence of a variety of higher-order membrane modes.

Table 1: 32-element CMUT Linear Array Properties

	Array
Number of elements	32
Element pitch	110 μm
Membrane pitch	55 μm
Element geometry	2×45
	Membrane (CMUT)
Shape	Square
Size	45×45 nm
Thickness	2.2 μm
Isolation thickness	200 nm
Gap	47 nm
Electrode coverage	100 %
Center frequency	6.3 MHz (in water)

Next, the array was simulated with a phased excitation of all the elements at a focus 3 cm from the center of the array. The pressure response magnitude for each node density is plotted in Fig. 28. The cross-talk effects observed in the previous case are smoothed out due to forcing of all the membranes. Above 18 MHz (about 3 times the single membrane fundamental), we see that the prediction for the first anti-resonance, the second resonance peak, and the overall shape of the response vary significantly depending on the node density. We begin to see convergence of the solutions with a sampling length of $w/8$ and below. A 2-D image of the mean membrane displacement magnitude at 5.5 MHz for the phased case is shown in Fig. 29. Strong periodicity is observed in the array modes in both dimensions, indicating that the finiteness of the array is involved in determining the overall array behavior due to the standing evanescent waves.

4.1.2 Channel cross-talk

When one channel (i.e. element) is excited, cross-talk levels on the other channels provide insight into the amount of undesireable interaction one can expect when the array is phased for imaging. To simulate this case, the right-most channel (CH31) was excited with a uniform voltage and the mean velocity of the three nearest channels (CH30, CH29, and CH28) were observed (see Fig. 32). Also included is the mean velocity of the left-most and farthest channel (CH0). Two regions of interest (ROI) are highlighted: (1) the 3-7 MHz region associated with cross-coupling of the first membrane

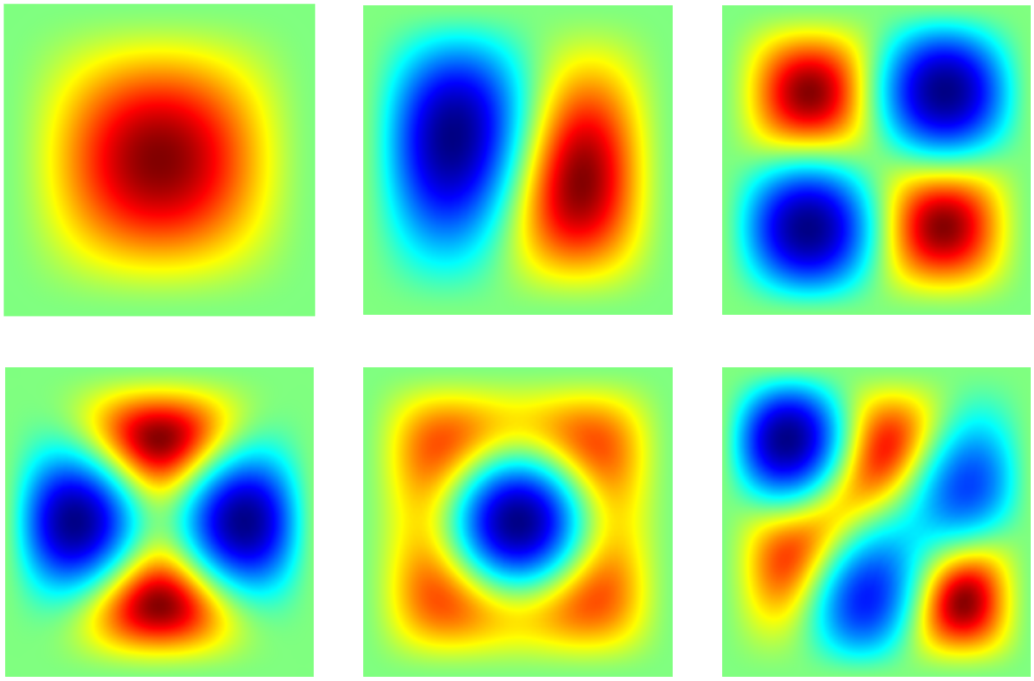


Figure 27: First six resonant mode shapes of the clamped square membrane.

mode (see Fig. 36), and (2) the 14-17 MHz region associated with cross-coupling of higher-order modes (the sixth mode in particular). Between the two regions, a usable band can be defined in which the channel cross-talk levels are reduced.

4.1.3 Element directivity

When a single element of the array is excited, the directivity of the element will be undesireably affected by the motion of its neighbors. It is expected that element directivity will change drastically depending on the relative position of the element in the array and on the frequency of excitation. In Fig. 33, the directivity of the array is plotted at different frequencies for excitation of an element near the array center (CH15) and excitation of the right-most element (CH31). Two reference cases are also provided: directivity of a single element assuming uniform piston motion and the absence of fluid-structure coupling, and directivity of a single element in isolation, i.e. without neighboring elements.

In ROI 1 (6.3 MHz), directivity is heavily compromised by the excitation of array modes associated with coupling of the first membrane mode. These array modes have shorter wavelengths on the order of the array pitch, generating the rapidly oscillating interference patterns shown. A significant asymmetry is observed, reflecting the fact that the excited element is not positioned symmetrically in the array. The close match between the two reference cases can be explained by the fact that

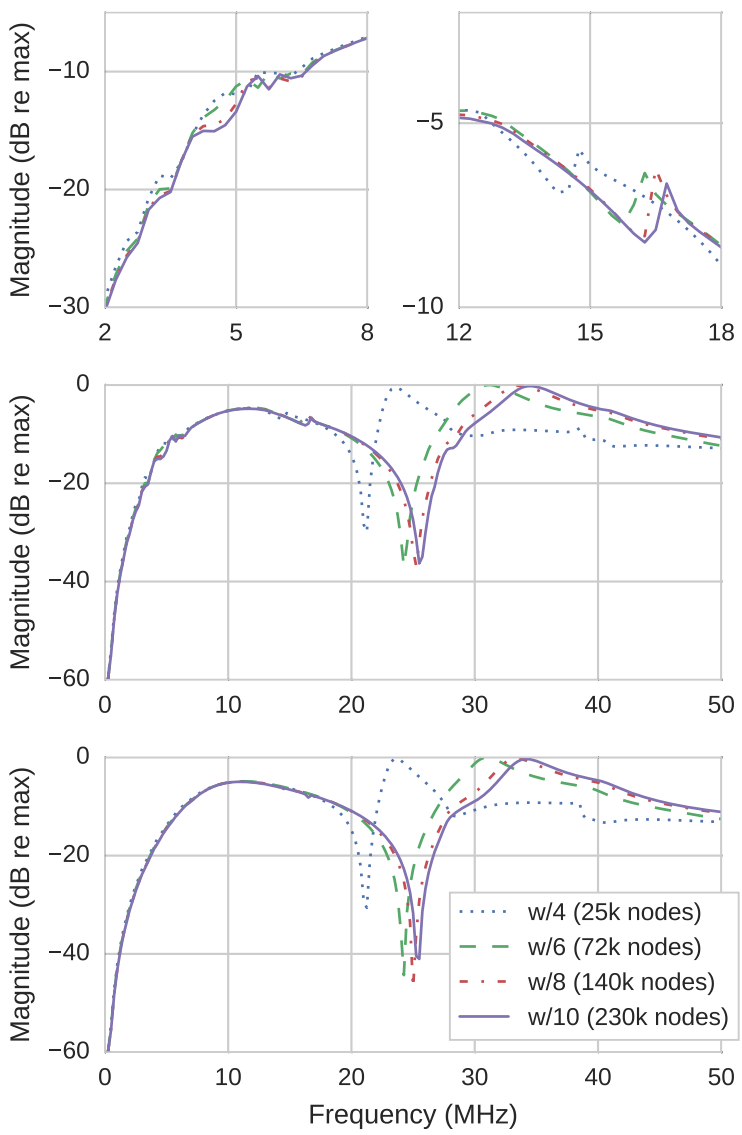


Figure 28: Pressure magnitude at 3 cm from the center of the array for two excitation cases. The simulation was performed for decreasing node sampling lengths where w is the membrane width. Top: Only the first element of the array is excited. Strong cross-talk is observed in the 3 - 7 MHz and 14 - 17 MHz bands (close-up is shown). Center: Full response for the same case. Bottom: All 32 elements are phased.

the first membrane mode shape involves large in-phase motion of the membrane, similar to that of a uniform piston.

In the usable band (10 MHz), the directivity is well-behaved due to the reduced cross-talk levels. The directivity of the CH31 again exhibits a significant asymmetry because it is surrounded on the right by a hard baffle, and on the left by a softened baffle due to the presence of neighboring membranes.

In ROI 2 (16.3 MHz), the directivity again is compromised the excitation of array modes, this time those associated with coupling of higher-order membrane modes. Here, the reference cases diverge as

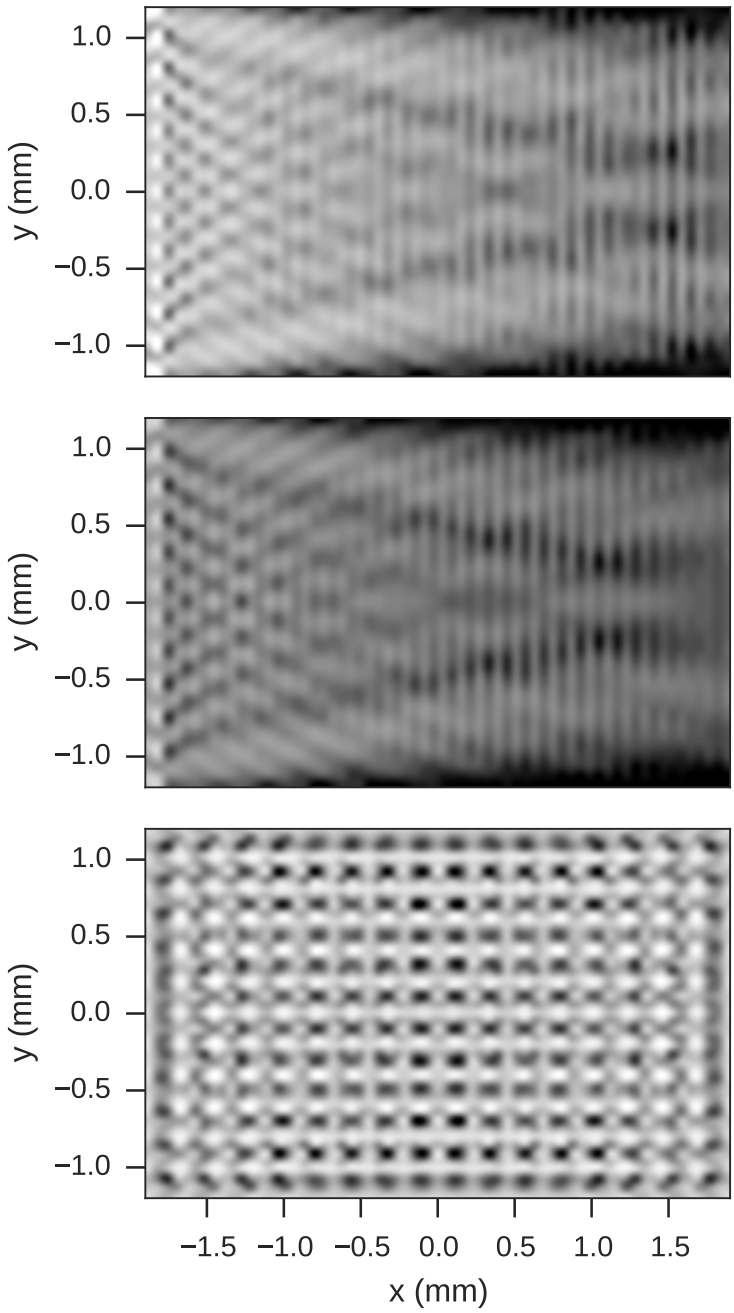


Figure 29: 2-D images of the mean membrane displacement at 5.5 MHz. The images are plotted on a log scale with 25 dB dynamic range. Note that the spaces between the membranes have been removed for these plots. Top: Array mode when only the first element is excited, simulated using the coarsest mesh ($w/4$). Center: Array mode when only the first element is excited, simulated using the finest mesh ($w/10$). Bottom: Array mode when all elements are phased, simulated using the finest mesh ($w/10$).

the higher-order membrane modeshapes are no longer well-approximated by uniform piston motion.

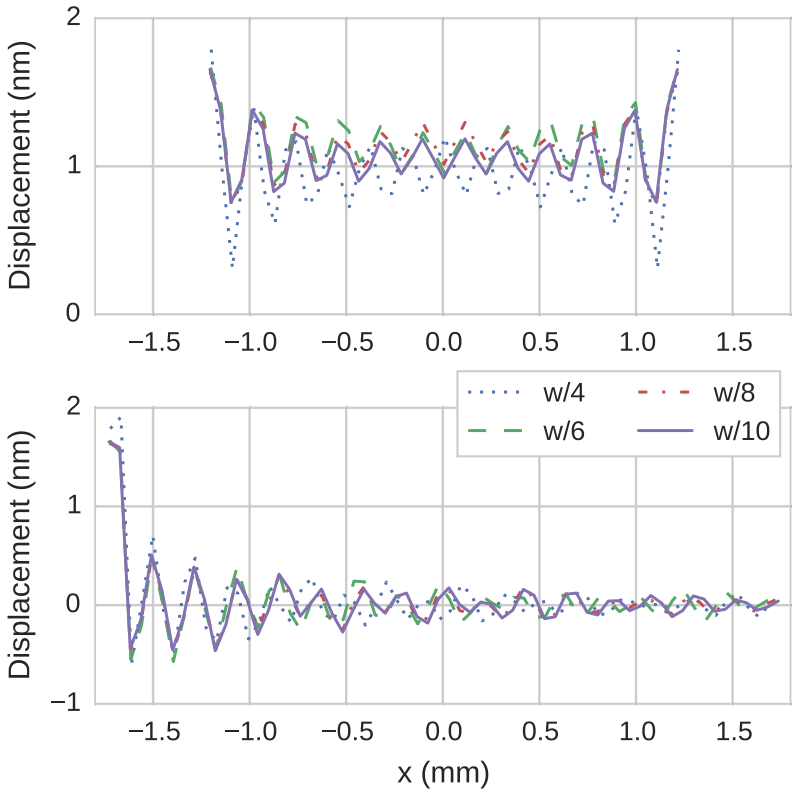


Figure 30: Displacement profiles of the array at 5.5 MHz, constructed from the displacement of the center node of each membrane, for the case of excitation of the first element only. Top: The profile of the leftmost column of membranes which are all excited. Bottom: The profile of the center row of membranes where the first two membranes are excited.

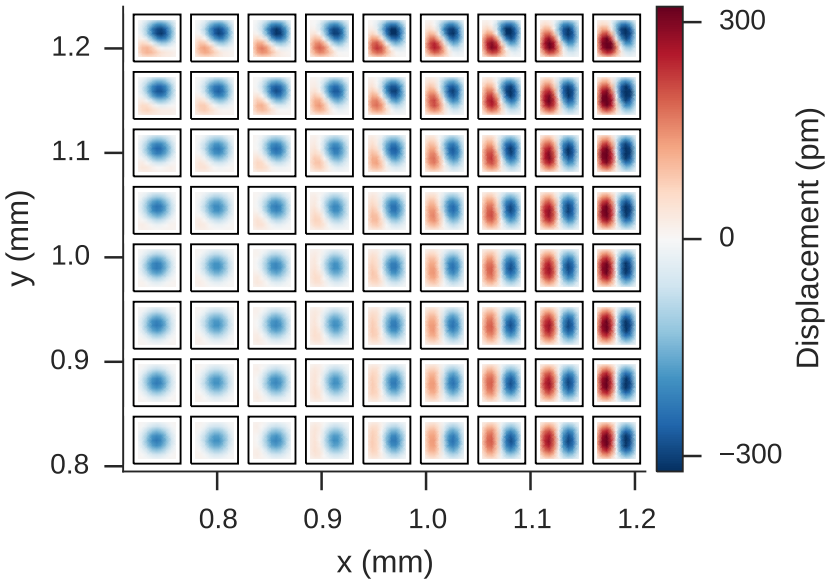


Figure 31: The membranes of the 1-D CMUT array at a frequency of 16.5 MHz are excited into a variety of higher-order modes due to cross-talk over the entire array.

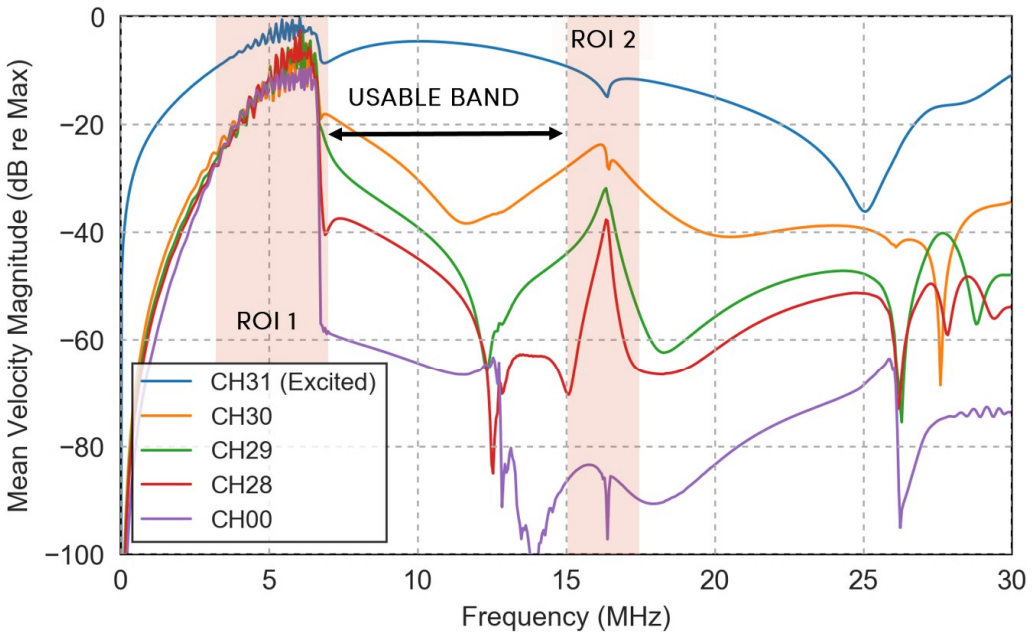
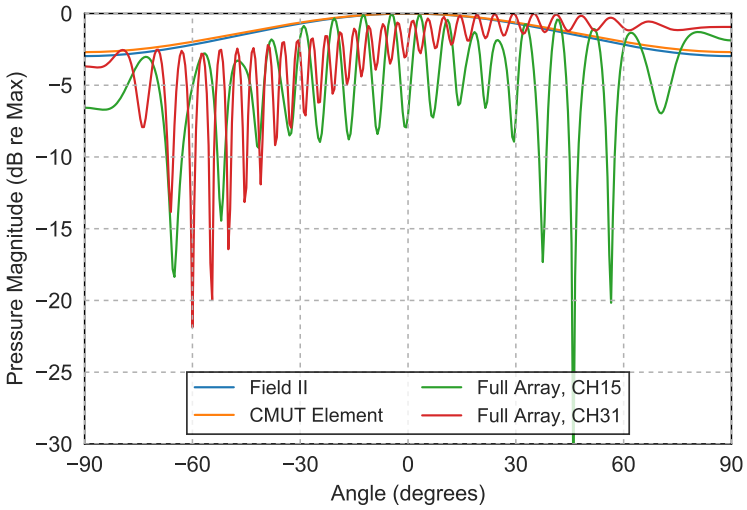


Figure 32: Mean velocity of neighboring channels when the right-most channel (CH31) is excited.

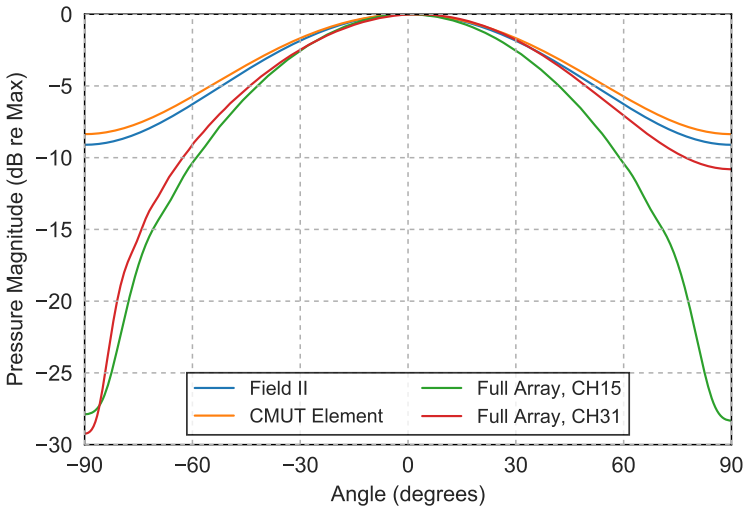
4.1.4 Bandwidth broadening

It is well-documented that CMUT array center frequency and bandwidth will differ significantly from single membrane characteristics as a result of fluid-loading. This is a consequence of a radiation impedance which changes with the effective surface area. A classic example of this behavior is the uniform circular piston which has a radiation resistance and reactance that depends on the ratio of the radius to the wavelength. As the piston radius increases, the reactance gradually vanishes and the resistance becomes approximately constant with respect to frequency, effectively broadening the bandwidth as a result.

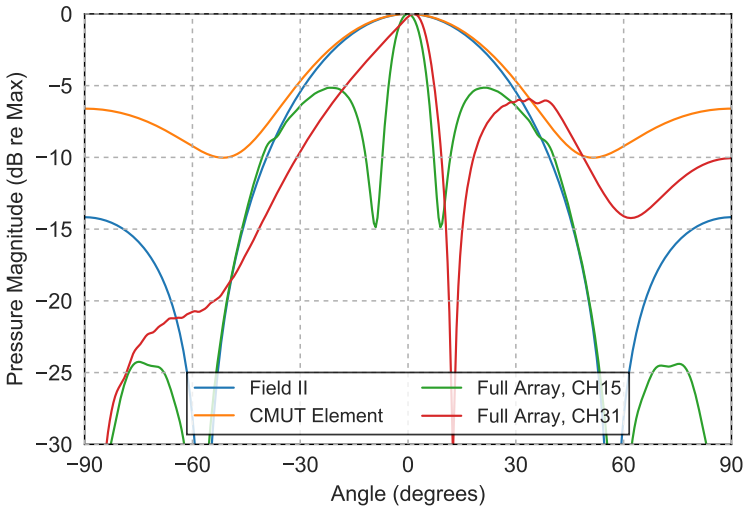
A similar behavior is exhibited in CMUT arrays with membranes that are densely packed. To study this broadening effect more closely, the pressure in front of the CMUT array was calculated for excitations involving a gradually increasing number of membranes (i.e., a gradually increasing effective surface area). This is shown in Fig. 34. When a single membrane is excited, despite the presence of the array, the response is very narrowband with center frequency corresponding to the single membrane resonance. As the number of excited membranes increases, the response around 10-12 MHz gradually increases, surpassing the level of the single membrane resonance somewhere



(a) Directivity at 6.3 MHz (ROI 1)



(b) Directivity at 10 MHz (usable band)



(c) Directivity at 16.3 MHz (ROI 2)

Figure 33: Element directivity for frequencies corresponding to different bands of the transducer.

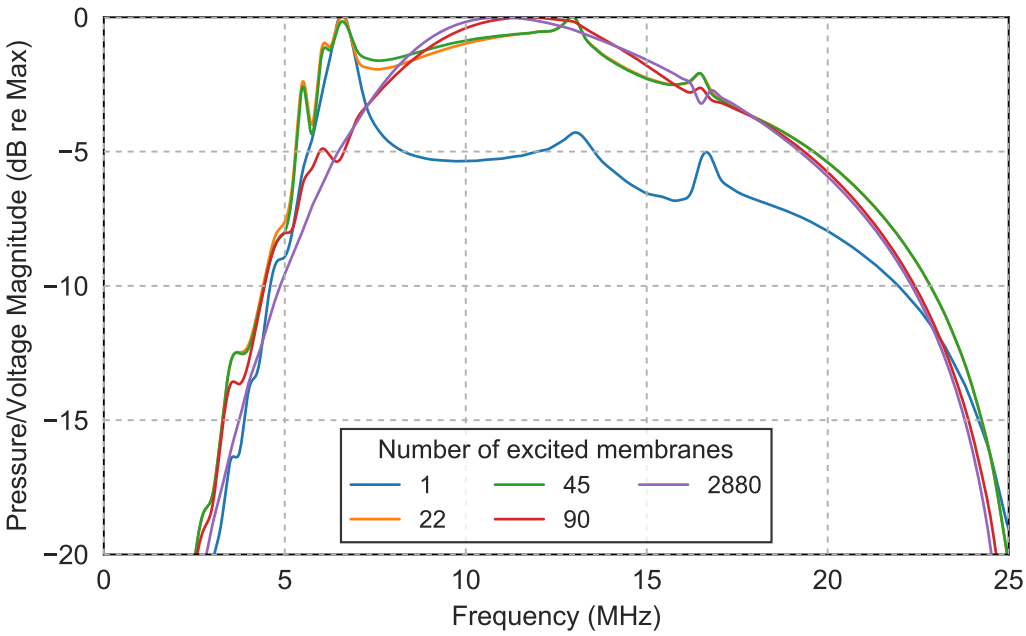


Figure 34: Pressure response as a function of the number of excited membranes.

between 45 and 90 membranes. At this point, the center frequency shifts upwards to 12 MHz and the bandwidth widens to its asymptotic value. Unlike the circular piston, the bandwidth of the fundamental band of the CMUT array will always be limited at high frequencies by the involvement of higher-order membrane modes.

4.2 7 by 7 PMUT matrix array

A 7 by 7 PMUT matrix array with circular membranes was simulated. The simulated membranes consisted of a 1 μm thick lead zirconate titanate (PZT) layer deposited on a 2.2 μm silicon oxide layer, both extending across the entire membrane. For simplicity, the default material properties from COMSOL were used (specifically, bulk PZT-5H and isotropic SiO_2 with $\rho_m = 2200\ kg/m^3$, $Y_m = 80\ GPa$ and $\nu_m = 0.17$). Top electrode coverage was set to 70% of the membrane diameter. This membrane was found to have a first resonance in water at $f_o = 8.75$ MHz. Array pitch was 55 μm, corresponding to an area fill-factor of 53%.

4.2.1 Center and full array excitation

The response of the array was investigated for the cases of excitation of the center membrane only and excitation of all membranes. The mean displacement response of the center membrane in each case is plotted as a function of frequency in Fig. 37. It is apparent that significant array

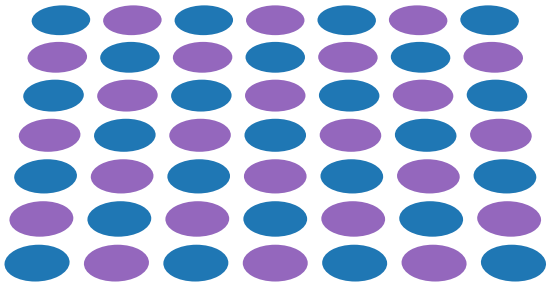


Figure 35: 7 by 7 PMUT matrix array.

modes (dispersive-guided modes) are excited in the 8 - 9 MHz range due to cross-coupling of the first membrane mode. For an array pitch of 55 μm $< \lambda_o/2$, where $\lambda_o = 171$ μm is the wavelength associated with the first membrane resonance f_o, both subsonic trapped modes (with associated phase speed below c) and leaky supersonic modes (with associated phase speed above c) are supported by the array [56]. Broadening of the bandwidth associated with increased aperture size is also observed when comparing center and full excitation cases. A secondary array mode is excited around 19 MHz corresponding to coupling of a higher-order mode.

4.2.2 Effect of array pitch

It has been shown that cross-talk phenomena is strongly contingent on membrane size and spacing which can be useful parameter for optimization [60, 109, 110]. Simulations were performed to investigate these behaviors in PMUT matrix arrays. First, simulations were performed for a 7 by 7 matrix array with pitches d of 0.3, 0.4, 0.5, 0.6, 0.8 and 1.0 of λ_o. The corresponding area fill factors were 60%, 34%, 22%, 15%, 8%, and 5%, respectively. In each case, the entire array was excited and the resulting mean displacement response plotted (see Fig. 38a).

A number of notable phenomena are observed. For pitches of $0.3\lambda_o$ and $0.4\lambda_o$, both subsonic and supersonic array modes are excited in a small region around the single membrane resonance, resulting in large resonant peaks and dips around f_o. In comparison with the single membrane response, the center frequency shifts upward and the overall bandwidth increases. For imaging applications, it will be typical to operate in this region for the broad bandwidth and fulfillment of the $\lambda/2$ Nyquist criterion. As the array pitch increases, the wavelengths associated with the same array modes lengthen, increasing their associated phase speeds. When the membrane spacing exceeds $\lambda_o/2$, these phase speeds fall above c, and the array modes no longer appear as resonant peaks and dips in the response. Additionally, an increase in the peak displacement and narrowing of the

Table 2: 7 by 7 PMUT Matrix Array Properties

	Array
Number of elements	49
Element pitch	varies
	Membrane (PMUT)
Shape	Circle
Radius	22.5 nm
Piezoelectric	1 μm PZT
Elastic	2.2 μm SiO_2
Electrode coverage	70%
Center frequency	8.75 MHz (in water)

bandwidth is observed with peak displacement occuring around $d = 0.8\lambda_o$. This suggests that, for narrowband applications, output pressure may be maximized by a sparse arrangement of membranes, as opposed to a tightly-packed arrangement, assuming the active surface area remains the same in both cases. Eventually, as the membrane spacing tends to infinity, it is expected that the membranes will behave independently and the mean array displacement response will approach that of a single membrane. The asymptotic behavior of the response is verified with a pitch of $10\lambda_o$.

It has been suggested that total radiation resistance may be a useful parameter for optimizing power efficiency of transducer arrays [109]. Total normalized radiation resistance, defined as the real part of the sum of the matrix elements of $\mathbf{Z}_r$ normalized by ρcS where S is the total active surface area, is investigated as a function of normalized parameters. The resistance is plotted as a function of frequency for constant pitches d/a where a is the membrane radius (Fig. 38b) and for different array sizes with constant pitch $d/a = 4$ (Fig. 38c). Parameter trends are found to behave similarly to that for plane baffled piston radiators [109] and CMUTs [60], but with additional complexity due to membrane flexure, square grid membrane arrangement, and support for non-axisymmetric membrane motion. Hybrid BEM can be a powerful tool for comprehensive exploration of these parameter spaces.

In Fig. 40a, the topography of the array is depicted at 8.75 MHz for harmonic excitation of all membranes. Here, the effect of acoustic cross-talk is shown clearly to enhance and/or suppress the average displacement of certain membranes depending on their geometric relation within the array. Furthermore, deflection profiles at zero phase, obtained along x-directed slices of the membranes, reveals complexity of the deflection beyond vibration of the first membrane mode. The normalized

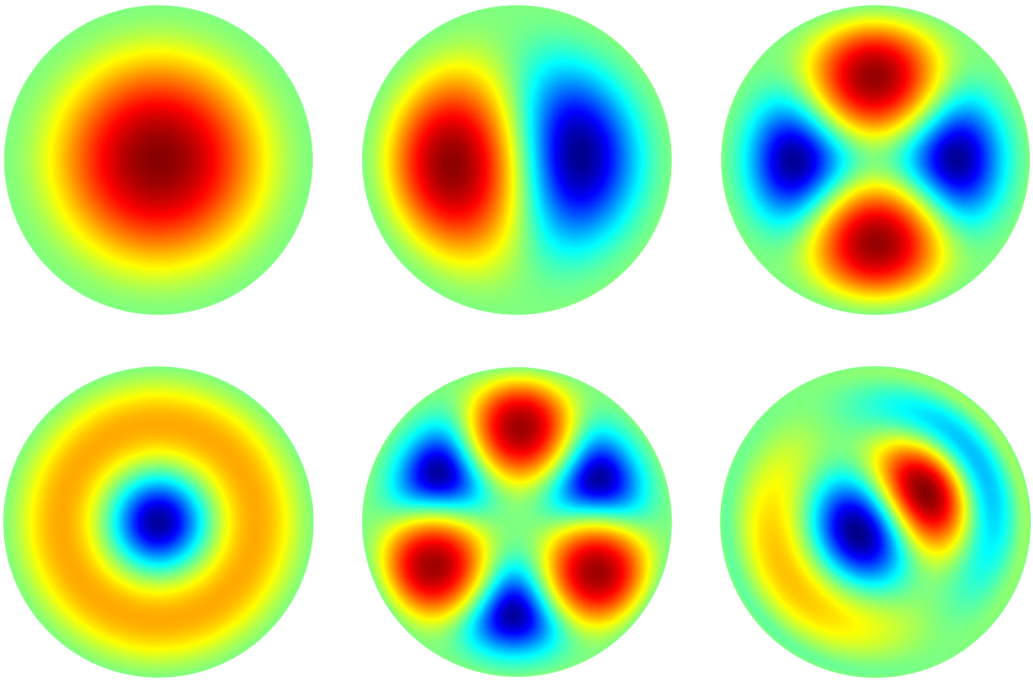


Figure 36: First six resonant mode shapes of the clamped circular membrane.

profiles for all 49 membranes, shown in Fig. 40b, indicate contribution from higher-order modes. This shows conclusively that membrane motion, even when excited in an axisymmetric manner, may exhibit non-axisymmetric motion due to the complex acoustic interaction between membranes.

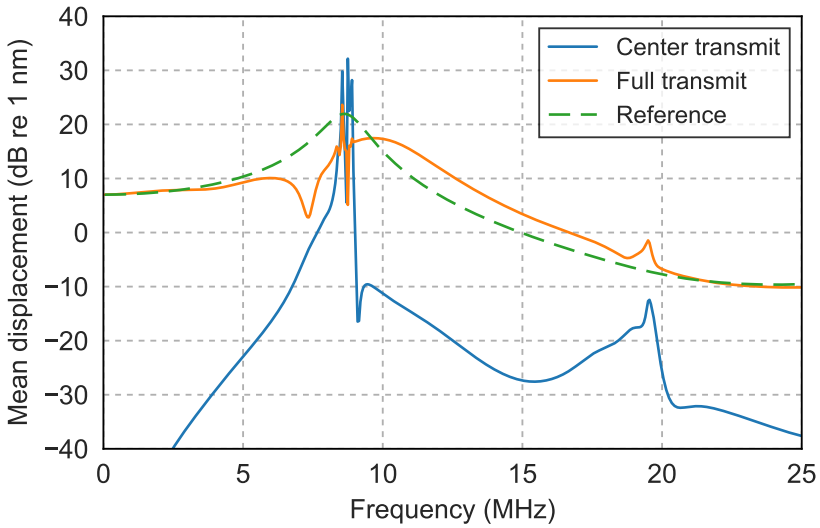


Figure 37: Mean displacement of the center membrane of a 5 by 5 matrix array when only the center membrane is excited, and when all membranes are excited in phase. Reference case shown is for an isolated membrane (without neighbors).

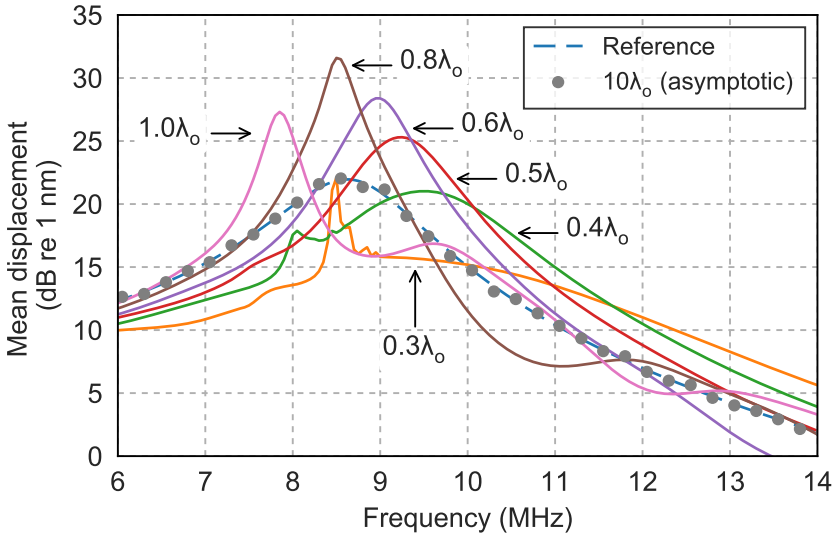
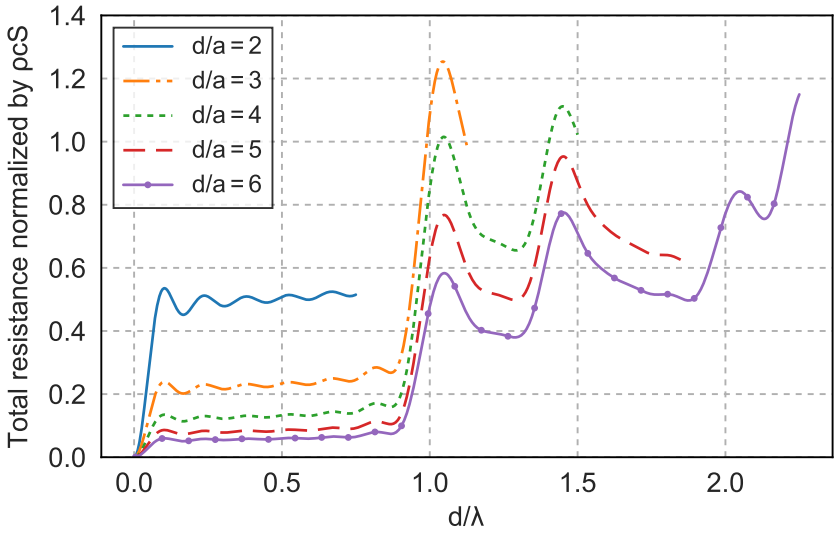
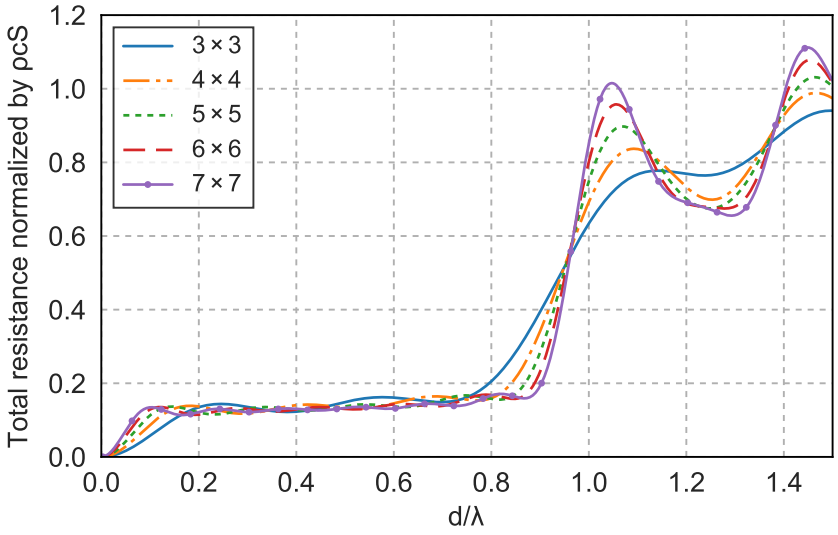


Figure 38: Mean displacement as a function of increasing array pitch.

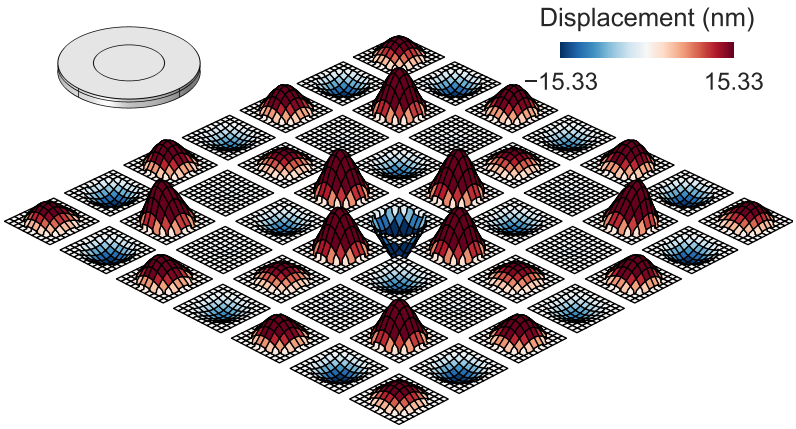


(a)

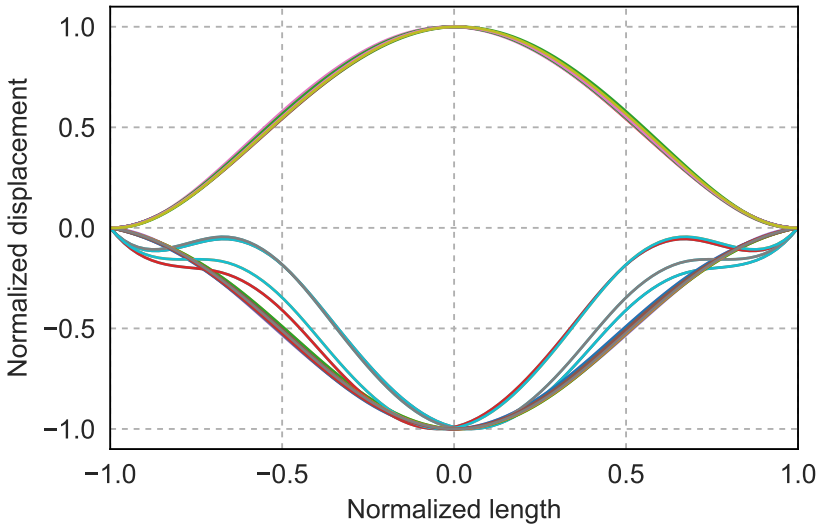


(b)

Figure 39: (a) Total radiation resistance normalized by ρcS for active surface area S as a function of frequency for a 7 by 7 matrix array and various values of d/a (pitch to radius ratio). (b) Total normalized radiation resistance as a function of frequency for fixed $d/a = 4$ and various matrix array sizes.



(a)



(b)

Figure 40: (a) Surface topography of a 7 by 7 matrix array at 8.75 MHz. (b) Displacement slices at the same frequency which indicate contributions from higher-order and axi-symmetric membrane modes.

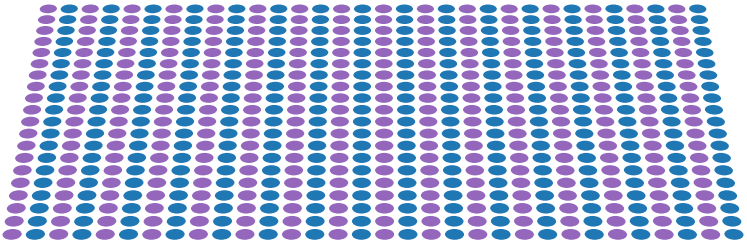


Figure 41: 32-element PMUT linear array.

4.3 32-element PMUT linear array

As an example of practical optimization of realistic arrays, such as those one may find in diagnostic imaging applications, a very large 32-element linear PMUT array with 640 membranes was simulated. The circular membranes (45 μm diameter) consisted of a 1 μm piezoelectric layer deposited on a 2.2 μm silicon layer. Several common piezoelectric thin-film materials were considered and is specified for each simulated case (see Table 4). Material properties for aluminum nitride (AlN) [111, 112], zinc oxide (ZnO) [111, 113], and textured PZT [114, 115] were obtained from literature. Since mechanical properties of PZT thin-films are not well-characterized, elasticity values of bulk PZT were used instead. Top electrode coverage was set to 70% of the membrane length. Each elements of the array consisted of a single column of 20 membranes. Computation time for each simulation was approximately 5 hrs running on 20 CPU threads at 2.3 GHz.

4.3.1 Channel cross-talk

Channel-to-channel cross-talk was characterized by simulating the response of the array for 1 V excitation of the first element (CH0) only. The membranes (AlN/Si) had a resonant frequency of $f_o = 15.10$ MHz. In Fig. 42, the mean displacement response of select elements is plotted as a function of frequency. Here it is observed that the cross-coupling increases sharply as a function of frequency, reaching a peak around 12.8 MHz due to the excitation of strong array modes. The cross-talk between channels is very significant–within 10 to 20 dB below the level of the excited channel–even for channels located at the center (CH16) and at the opposite end (CH31) of the array.

Table 3: 32-element PMUT Linear Array Properties

	Array
Number of elements	32
Element pitch	55 μm
Membrane pitch	55 μm
Element geometry	1×20
	Membrane (PMUT)
Shape	Circle
Radius	22.5 nm
Piezoelectric	1 μm AlN/ZnO/PZT
Elastic	2.2 μm Si
Electrode coverage	varies
Center frequency	varies

4.3.2 Electrode optimization

Optimization of top electrode coverage, which can have a significant effect on the total output pressure (for examples, see [105, 106, 116]), is demonstrated. Simulations were performed for electrode coverage from 10% to 90% of the membrane radius (corresponding to 1% to 81% area coverage, respectively). All membranes (AlN/Si) were excited in phase and the output pressure response was calculated at a distance of 2 mm. The maximum output pressure (normalized) over the first band is plotted as a function of electrode coverage in Fig. 43 for the cases of full array simulation and single membrane simulation. In both cases, maximum output pressure is obtained for approximately 65% electrode coverage which matches reasonably with the 60% figure of [116].

The simulations suggest that optimal electrode coverage depends primarily on the ability of the electrode shape to couple motion into the first membrane mode. An analytical optimization curve can be derived based on a normal mode solution of the equation of motion for thin membranes (47) in cylindrical coordinates. Consider the piezoelectrically-induced loading function $p_{piezo}(r, \theta)$ in cylindrical coordinates for a top electrode with radius r_e

$$\begin{aligned} p_{piezo} &\propto \nabla^2(H(r + r_e) - H(r - r_e)) \\ &\propto \frac{1}{r}\delta(r + r_e) + \delta'(r + r_e) - \frac{1}{r}\delta(r - r_e) - \delta'(r - r_e) \end{aligned} \tag{55}$$

In a normal mode solution, the response is represented by a linear superposition of the mode shapes

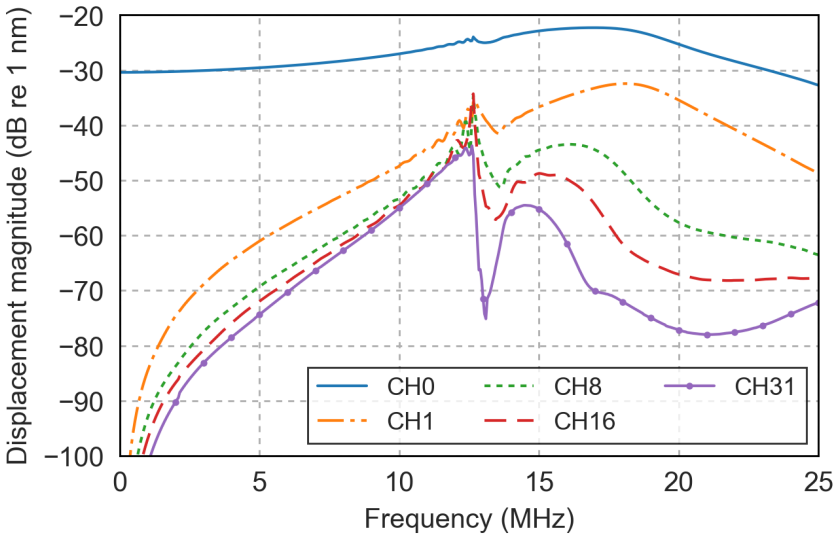


Figure 42: Mean displacement of selected elements of a 32-element linear array for the case of excitation of the first element only.

$\phi_n(r, \theta)$, obtained by solving the homogeneous equation. The contribution of each mode is determined by the projection of the load onto the mode shape. For the first mode, then

$$\langle p_{piezo}, \phi \rangle = \int_0^{2\pi} \int_0^R p_{piezo} \phi_1 \, r \, dr \, d\theta \tag{56}$$

With the following property

$$\int_{-\infty}^{\infty} \delta^{(n)}(x - y) f(x) \, dx = (-1)^n f^{(n)}(y) \tag{57}$$

and the fact that ϕ_1 is axisymmetric, it can be concluded that

$$\langle p_{piezo}, \phi \rangle \propto r_e \phi'(r_e) \tag{58}$$

The analytical curve, plotted in Fig. 43, predicts an optimal electrode coverage of about 67%, the difference which can be attributed to violation of the thin membrane condition, neglect of fluid-loading in this analytical approach, and possible contributions from higher-order axisymmetric modes.

A simple design intuition can be gleaned from this result: the electrode boundaries, which dictate the location of the applied bending moment, should be selected to maximize deflection of the desired membrane mode. The analytical curve suggests that the optimal location occurs when the product of the electrode radius and the derivative of the mode shape at that radius is maximized. When

Table 4: Thin Film Properties

	Si	AlN [111, 112]	ZnO [111, 113]	PZT [114, 115]
ρ (kg/m^3)	2329	3512	5720	7500
c_{11} (GPa)	166	345	209	114
c_{12} (GPa)	64	125	121	58
c_{13} (GPa)	64	120	105	59
c_{33} (GPa)	166	395	211	98
c_{44} (GPa)	80	118	42	21
c_{66} (GPa)	80	110	44	26
e_{31} (C/m^2)	-	-0.58	-0.57	-8
e_{33} (C/m^2)	-	1.55	1.32	6
e_{15} (C/m^2)	-	-0.48	-0.48	15
ϵ_{33}	-	11	11	1300

generalized to optimization of multiple electrode designs, the model also suggests that the polarity of neighboring electrodes should alternate to ensure that their associated moments add together. Hybrid BEM can supply additional fine tuning for this type of optimization by accounting for effects not included in the analytical model. Additionally, hybrid BEM provides a method for the numerical calculation and visualization of the piezoelectrically-induced loading function for complex electrode geometries.

4.3.3 Material optimization

Optimization of material choice was investigated by considering the pressure response for phased excitation at 2 mm, shown in Fig. 44. To ensure a fair comparison, membranes with AlN and ZnO layers were thinned while maintaining a constant piezoelectric/elastic thickness ratio (h_p/h_m) of 0.45 in order to match the resonance frequency of PZT/Si. The resulting simulated geometries were: AlN/Si ($f_o = 10.85$ MHz, $h_m = 1.67$ μm, $h_p = 0.76$ μm), ZnO/Si ($f_o = 10.80$ MHz, $h_m = 1.91$ μm, $h_p = 0.87$ μm), and PZT/Si ($f_o = 10.75$ MHz, $h_m = 2.20$ μm, $h_p = 1.00$ μm). Simulation indicates that pressure performance for PZT/Si is over 26 dB better than AlN/Si and over 21 dB better than ZnO/Si. Center frequency and bandwidth are nearly identical in all three cases, with small fractional

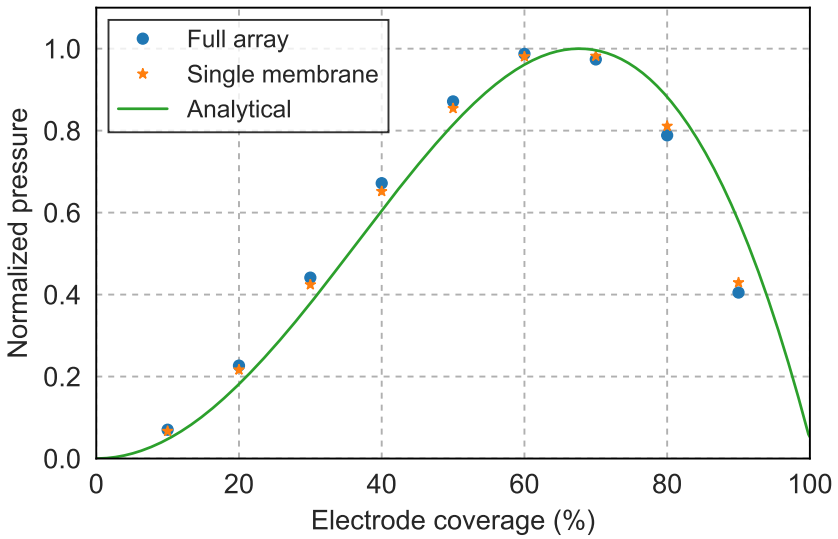


Figure 43: Maximum output pressure (normalized) in the first band as a function of electrode coverage for full array simulation, single membrane simulation, and analytical optimization from the thin-plate model.

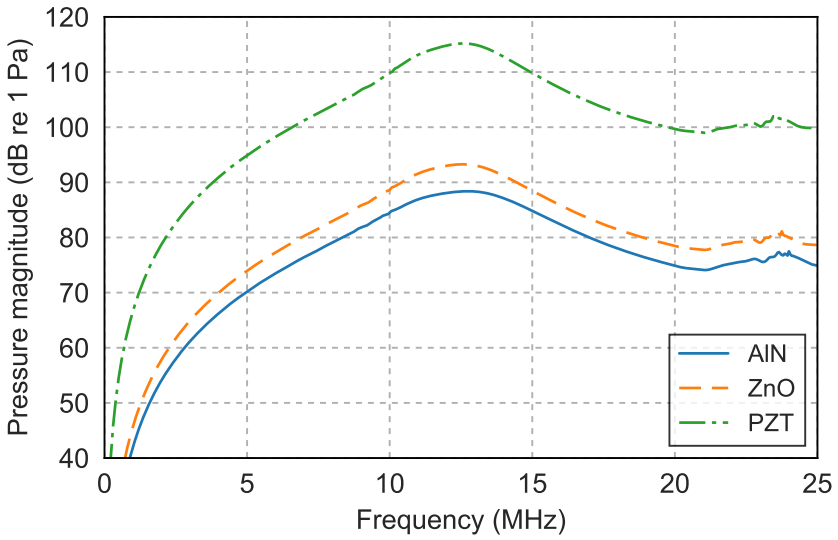


Figure 44: Frequency response for AlN/Si, ZnO/Si, and PZT/Si PMUTs.

bandwidth improvements over PZT/Si (29%) for AlN/Si (33%) and ZnO/Si (37%).

4.4 Summary

We have presented several examples of large array simulation and optimization. A 32-element CMUT linear array was simulated for the purpose of investigating cross-talk phenomena such as channel cross-talk, element directivity, and bandwidth broadening. A 7 by 7 PMUT matrix array was simulated understanding the effect of array pitch on pressure output and radiation resistance. Material and electrode coverage optimization were explored through simulation of a 32-element linear array with 640 membranes.

CHAPTER 5

IMAGING PERFORMANCE ANALYSIS OF A FOLDABLE LARGE APERTURE 2-D ARRAY FOR INTRACARDIAC ECHOCARDIOGRAPHY

5.1 Background and motivation

For intracardiac echocardiography (ICE) to experience continued adoption in the catheter lab, it must remain competitive with recent advances in transesophageal (TEE) and transthoracic (TTE) echocardiography. State-of-the-art for both TEE and TTE now feature real-time volumetric (3D/4D) imaging modes enabled by large aperture arrays with several thousand elements. For example, the specifications of commercially-available probes from Philips (Amsterdam, Netherlands) and General Electric (Boston, MA, USA) are shown in Table 5.

ICE arrays, by virtue of constraints imposed by the catheter form-factor, have a difficult path forward towards achieving technical specifications on par with TTE and TEE state-of-the-art. Due to the limited catheter diameter, array size of a side-looking design is constrained in one dimension to about 10 Fr (3.3 mm diameter). Element quantity is also constrained, limited by the maximum number of cables which can be carried in the catheter interior, typically around 60–80 cables.

With the aim of bridging this performance gap, a foldable large aperture 2-D ICE array based on the CMUT platform is envisioned. The proposed design consists of multiple folding panes which, when collapsed, can be delivered via catheter to the heart. Once in the heart, the panes are deployed mechanically to form a large aperture, thereby superseding the size limit of the catheter. Further performance improvements are enabled by the CMUT platform and its electronic integration capability. Element quantity is increased by the use of on-chip solutions for transmit pulse formation, sub-aperture beamformation [117], and time-division multiplexing (TDM) [118]. Layout flexibility and precision from microfabrication facilitate the use of well-known sparse array designs which maximize performance from a limited number of elements.

While this is, to the best of our knowledge, the first instance in the literature of a foldable ICE array, the idea of a multi-part flexible array is not new. So-called flexible/conformal transducer arrays has been explored previously for non-destructive evaluation (NDE) and medical applications. For example, in [119], flexible arrays consisting of piezopolymer and piezocomposite transducers were considered for imaging solid structures with curved surface topography. In [120], a jointed structure composed of 24 piezoelectric elements was tested for ultrasonic inspection of solid components with

Table 5: TEE/TTE State-of-the-Art

	Philips		GE	
	X7-2t (TEE)	X5-1 (TTE)	6VT-D (TEE)	4V-D (TTE)
Number of elements	2500	3040	-	-
Aperture	-	-	13 mm	22 mm
Frequency range	2–7 MHz	1–5 MHz	3–8 MHz	1.5–4 MHz
Field of view	$98^\circ \times 98^\circ$	$98^\circ \times 98^\circ$	$90^\circ \times 90^\circ$	$90^\circ \times 90^\circ$
Penetration depth (max)	-	-	20 cm	30 cm

complex geometries. In the field of medical ultrasound, conformal arrays have been constructed for ultrasound therapy [121, 122] and imaging [123, 124]. Advanced studies have been carried out to correct for phase aberration issues when imaging with conformal arrays [125].

In this chapter, we analyze the imaging performance of the proposed foldable array design through simulation. Imaging performance is measured through calculation of standard measures, such as detail resolution, contrast resolution, and signal-to-noise ratio. The structure of this chapter is as follows. The proposed array design is described in detail and its target performance outlined. The imaging measures used to assess performance are defined. The simulation methodology is described, including the calculation of beamplots and simulated tissue phantoms. Imaging performance results are shown and analyzed. The problem of performance degradation due to orientation error is investigated. Finally, discussion and concluding remarks are given.

5.2 A three-pane foldable Vernier array design

A novel Vernier sparse array is considered here based closely on a design principle originally proposed by Lockwood et. al. [7, 126–128]. The array consists of the superposition of two arrays: a transmit array with elements arranged on a regular grid of pitch d_{tx}, and a receive array with elements arranged on a regular grid of pitch d_{rx}. In analogy with a sliding vernier scale (refer to Fig. 45), the transmit and receive arrays are spaced such that one scale is a constant fraction of the

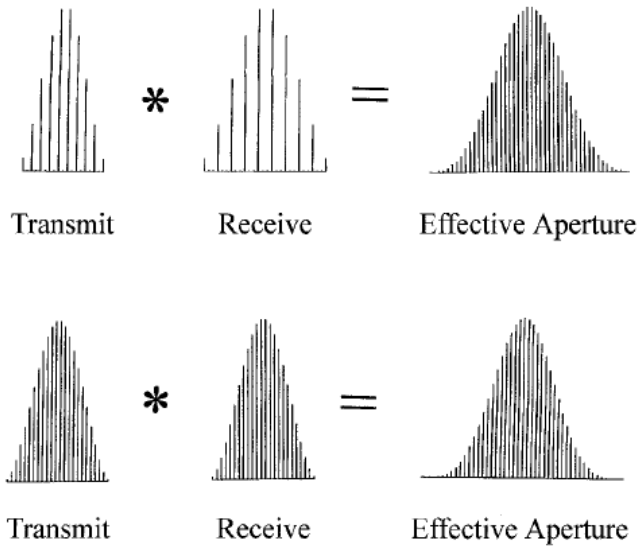


Figure 45: Vernier sparse array design principle. Copyright 1994 IEEE. Reprinted, with permission, from [7]

other. Specifically, the pitches must satisfy the following rule

$$p_{tx} = p * d \tag{59}$$
$$p_{rx} = (p-1) * d$$

where d is the design pitch and p is an integer parameter greater than 1. The effective aperture (also known as the co-array [90]), obtained from the 2-D spatial convolution of the transmit and receive arrays, will have pitch d which is also $1/p$ of p_{tx}. The design pitch should be small enough to satisfy the $\lambda/2$ spatial sampling criterion to avoid grating lobes. The integer p is a sparsity parameter, with larger values increasing the spacing of the array and the overall size.

A foldable 2-D array based on the Vernier concept is shown in Fig. 46. The array consists of three panes, each measuring 2.5 mm by 7.5 mm. It is assumed that the center pane is fixed to the catheter assembly and that the side panes are supported by a hinge mechanism which allows them to collapse onto the center pane during delivery. The array is composed of 517 transmit elements and 517 receive elements, 61 elements of which serve dual purpose as transmit and receive. The number of elements was determined conservatively based on the required transmit circuitry footprint and TDM capability. With 8-to-1 TDM, around 65 dedicated receive cables would be required. Element pitch was selected based on 7 MHz 80% fractional bandwidth (FBW) operation and a sparsity parameter of $p = 3$. Each element of the array is composed of a 2 by 2 grid of square CMUT membranes.

Performance targets, listed in Table 7, were selected based on current capabilities of commercial

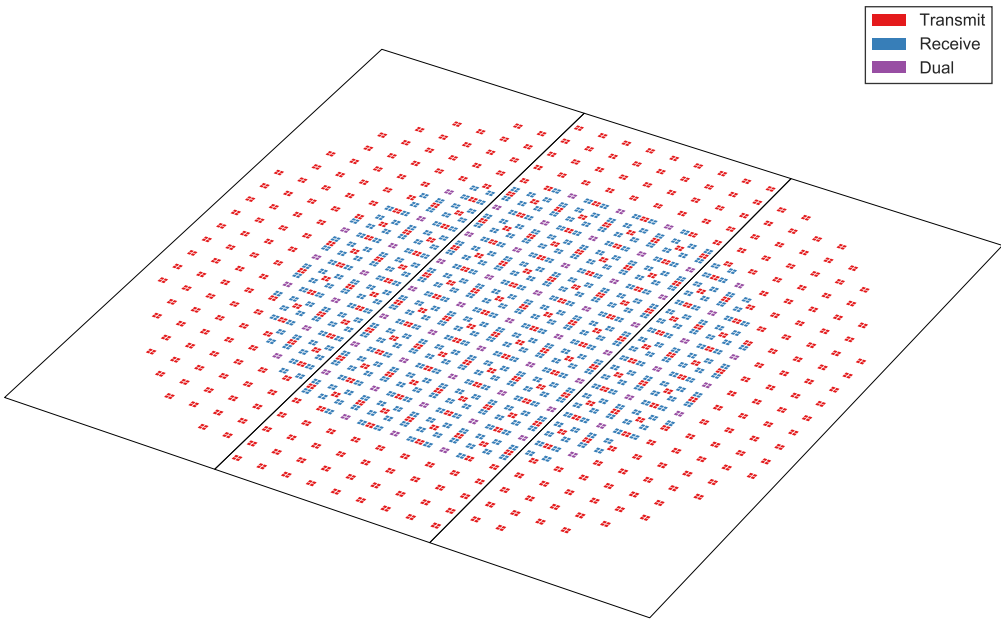


Figure 46: Foldable Vernier array design.

ICE probes. These performance measures are defined in the following section.

5.3 Imaging measures

To assess the imaging performance of the proposed ultrasound array, we adopt several standard performance measures which are defined here. The single most important measure is the **point spread function (PSF)** which characterizes the ultrasound system's response to an ideal impulse target. In normal operation (linear acoustics etc.), the resulting image of an object $o(\vec{r})$ is given by the spatial convolution product

$$Image(\vec{r}) = \int o(\vec{r}) PSF(\vec{r}_o - \vec{r}) d\vec{r} \tag{60}$$

The PSF therefore represents the impulse response of a three-dimensional linear spatial filter which smooths the object, resulting in a loss of detail. The **detail resolution**, defined as the minimum distance between two resolvable ideal point targets, is a measure of the level of detail retained in the image. In diffraction-limited systems like an ultrasound array, first-order estimates of the resolution can be derived by considering the array as a rectangular aperture with uniform normal velocity [90].

Table 6: Foldable Vernier array design

	Array
Number of elements	1034 (517 Tx/517 Rx)
Tx element pitch	$3(0.9)\lambda_o/2$
Rx element pitch	$2(0.9)\lambda_o/2$
Membrane pitch	45 μm
Element geometry	2×2
	Membrane (CMUT)
Shape	Square
Size	35×35 nm
Thickness	2.2 μm
Isolation thickness	200 nm
Gap	47 nm
Electrode coverage	100 %
Center frequency/FBW	7 MHz/80% (in water)

Table 7: Performance targets

Detail resolution	$<$ 1.72 mm
CTR	$<$ -10 dB
Penetration depth	12–15 cm

The resolution in the lateral dimensions is limited by the size of the aperture L and the wavelength λ,

$$d_{lateral} = FWHM = 1.206\frac{z\lambda}{L} \tag{61}$$

and in the axial dimension by the transducer bandwidth. For a more complete assessment, resolution should be determined by the lateral and axial main-lobe width of the PSF at the -6 dB (or sometimes -10 dB) level.

Sensitivity is a measure of the ability of an ultrasound system to detect echogenic signals. It is quantified by the **signal-to-noise ratio (SNR)**, where the noise reference level refers to noise sources originating from the ultrasound system itself (e.g., thermal-mechanical, electronic, and quantization noise).

$$SNR = 20log_{10}\left(\frac{B_{rms,signal}}{B_{rms,noise}}\right) \tag{62}$$

SNR can be measured experimentally by imaging pin-like targets at various locations, or by the use of a tissue-mimicking phantom. In either case, the noise level is measured by the rms brightness of an empty target area. In the following, we consider a system limited by thermal-mechanical noise, which has been demonstrated in CMUT systems employing very low-noise amplifiers [129]. An estimate of the thermal-mechanical pressure (the minimum-detectable pressure) is given by

$$p_{min} = \frac{\sqrt{\rho ckT \times Area \times Bandwidth}}{Area} \tag{63}$$

where k is the Boltzmann constant (1.38×10^{-23} J/K), T is the temperature in kelvin, ρ is the fluid density, and c is the fluid sound speed. Naturally, SNR will have a spatial dependence on depth due to path-dependent geometric and attenuation loss and on angle due to array directivity. The **penetration depth** refers to the depth at which the SNR drops to 6 dB.

Finally, **contrast resolution** refers to the ability to detect anechoic and weakly-echogenic objects in the presence of strong off-axis objects. Considering a spherical cyst located on-axis, acoustic clutter from off-axis objects will tend to fill in the cyst and decrease visibility. One measure for quantifying contrast resolution is the **clutter energy to total energy ratio (CTR)**, For a PSF concentric with a spherical volume (the cyst) of $5\lambda_o$ diameter, the CTR is the ratio of the integrated PSF energy outside the volume to the total integrated PSF energy.

$$CTR = 20log_{10}\left(\frac{\int_{r\notin V} PSF(\vec{r})d\vec{r}}{\int_r PSF(\vec{r})d\vec{r}}\right) \tag{64}$$

5.4 Methodology

5.4.1 Calculation of beamplots

Calculation of full-image PSFs is computationally intensive and requires knowledge of the entire imaging chain, i.e. transmit transduction, wave propagation/diffraction, receive transduction, envelope detection, post-processing etc. Since we are concerned primarily with optimization of array design and layout, we focus on the diffraction characteristics of the array which are most susceptible to these changes. Diffraction, governed by the principles of wave physics, is the primary limiting factor for detail and contrast resolution. To assess diffraction performance, we adopt the simpler two-way beamplot (sometimes referred to as the mechanical-PSF or the spatial impulse response), the characteristics of which provide a very good estimate of the main-lobe size and side-lobe levels

that can be expected in the full-image PSF.

To calculate beamplots, a linear systems model for acoustic propagation from apertures developed by Tupholme [130] and Stepanisen [131] provides an accurate and efficient simulation framework. We employ the open-source program Field II [132] which adopts this model. Additional modifications for element factor, backscatter, and path-dependent attenuation are included to enhance the realism of the simulation. The linear systems model and its modifications are described here.

Acoustic propagation and backscatter from transducers is treated in the linear systems model as a set of impulse responses. The time-domain description from front-face normal acceleration $a(t)$ to the mean received pressure $\bar{p}_r(t)$ is given by

$$\bar{p}_r(t) = \rho a(t) * h_{tx}(\vec{r}, t) \; * \frac{2\pi\sqrt{\sigma_{bs}}}{S_r} h_{rx}(\vec{r}, t) \tag{65}$$

Here $\vec{r}$ is the observation vector pointing to the target location in the field, ρ is the ambient density of the medium, σ_{bs} is the differential backscattering cross-section, S_r is the receive apertuer area, and tx and rx subscripts are used to distinguish between transmit and receive functions, respectively. $h(\vec{r}, t)$ is the spatial impulse response of the transducer defined as

$$h(\vec{r}, t) \equiv \iint_S \frac{\delta(t - \frac{|\vec{r} - \vec{r}_s|}{c})}{2\pi|\vec{r} - \vec{r}_s|} \, \mathrm{d}S \tag{66}$$

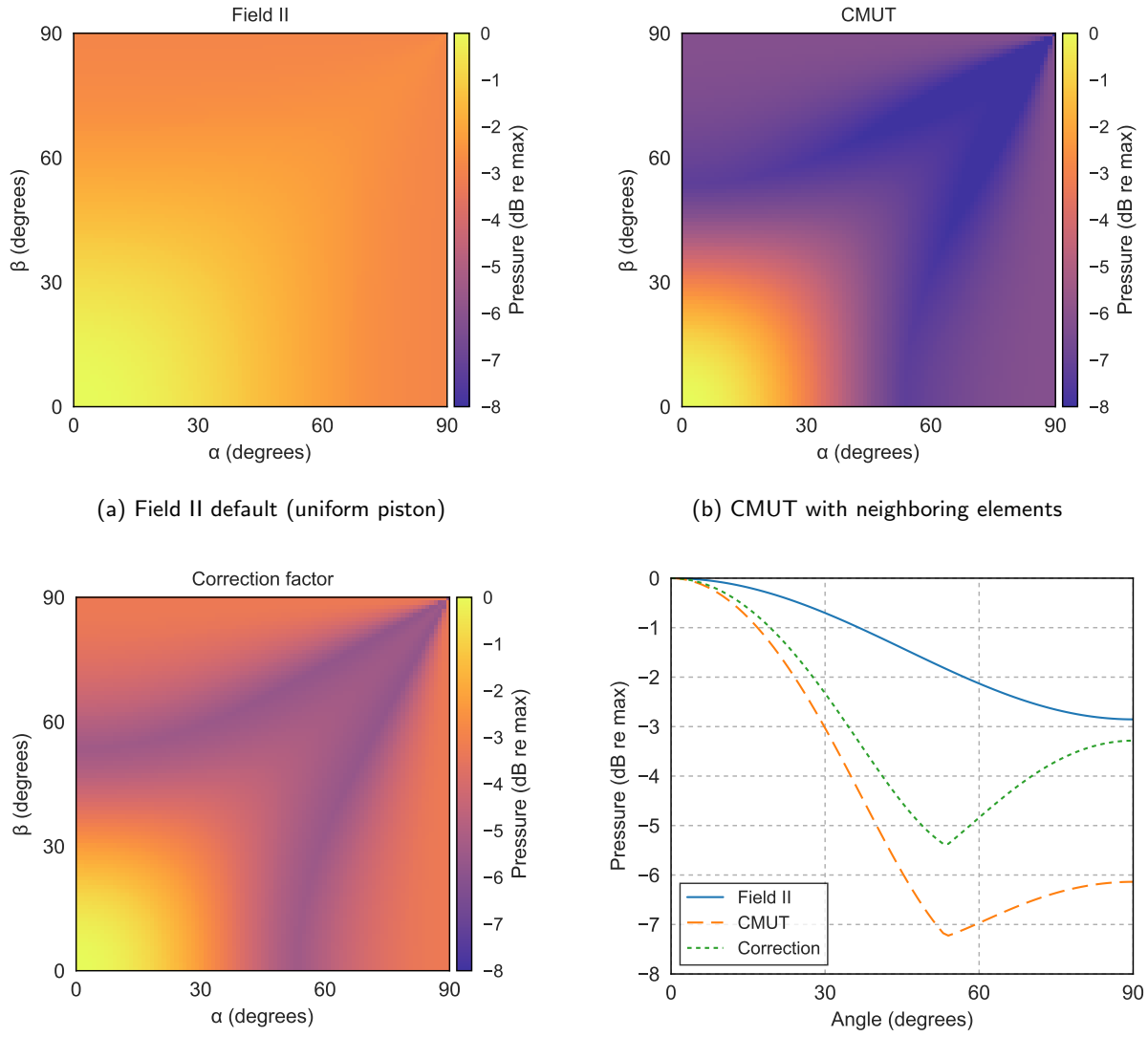
where $\vec{r}_s$ is the source vector pointing to the aperture surface, c is the sound speed, $\delta(t)$ is the Dirac impulse function, and the integration is carried out over the entire aperture surface. From the definition, the spatial impulse response is the summation of scaled and delayed impulses which originate from the aperture surface.

To gain a better intuition of (65), consider the grouping of certain terms. The first two terms yields

$$p_{in}(t) = \rho a(t) * h_{tx,n}(\vec{r}, t) = \rho \iint_S \frac{a(t - \frac{|\vec{r} - \vec{r}_s|}{c})}{2\pi|\vec{r} - \vec{r}_s|} \, \mathrm{d}S \tag{67}$$

which one will recognize as the Rayleigh integral for pressure from planar baffled sources. It is the incident pressure at location $\vec{r}$ due to the transmit aperture.

To understand the remaining terms, we consider a simple monopole scattering model for a target located at $\vec{r}$. The target scattering amplitude is described by σ_{bs}, which is related to the total scattering cross-section, defined as the acoustic source power of the scattered field per unit



(a) Field II default (uniform piston)

(b) CMUT with neighboring elements

(c) Correction factor

(d) Correction factor slice

Figure 47: Element factor correction for a CMUT element with neighboring elements.

incident intensity, by $\sigma_{bs}/4\pi = \sigma_t$. The backscattered field in the direction of the receive aperture at a location $\vec{r}_s'$ is given by

$$p_s(t) = p_{in}(t) * \sqrt{4\pi\sigma_{bs}} \frac{\delta(t - \frac{|\vec{r}-\vec{r}_s'|}{c})}{\sqrt{4\pi}|\vec{r} - \vec{r}_s'|} \tag{68}$$

To find the average pressure on the receive aperture, we integrate over the receive aperture and divide by the total receive aperture area S_r

$$\begin{aligned} \bar{p}_r(t) &= p_{in}(t) * \frac{\sqrt{\sigma_{bs}}}{S_r} \iint_S' \frac{\delta(t - \frac{|\vec{r}-\vec{r}_s'|}{c})}{|\vec{r} - \vec{r}_s'|} \, \mathrm{d}S' \\ &= p_{in}(t) * \frac{2\pi\sqrt{\sigma_{bs}}}{S_r} h_{rx}(\vec{r}, t) \end{aligned} \tag{69}$$

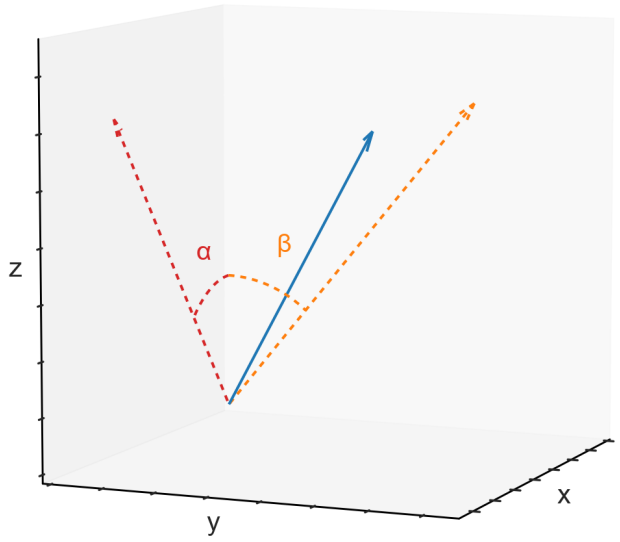


Figure 48: Coordinates used in the generation of beamplots.

which explains the remaining terms of (65)

So far, we have assumed a uniform normal acceleration profile over the surface of the transducer. In order to represent phased and apodized apertures, the array surfaces are discretized into smaller mathematical elements which can be individually phased and weighted as desired. Each mathematical element will also have its own transmit and receive spatial impulse response functions. For example, consider the division of the the transmit and receive apertures into N and M mathematical elements, respectively. Then,

$$\bar{p}_r(t) = \rho \sum_n \left[w_n a_n(t) * h_{tx,n}(\vec{r}, t - \tau_n) \right] * \frac{2\pi\sqrt{\sigma_{bs}}}{S_r} \sum_m \left[w_m h_{rx,m}(\vec{r}, t - \tau_m) \right] \tag{70}$$

where w_n is the weighting and τ_n is the phase delay of the n-th element.

Proper selection of apodization weights, which act as a smoothing kernel for the beamplot response, is important for suppressing side-lobe levels at the expense of a small increase in main-lobe width. We consider here a standard cosine weighting function based on the distance r of an element from the center of the array and the total radius of the aperture R

$$w(r) = cos\left(\frac{\pi r}{2R}\right) \tag{71}$$

Element factor refers to the natural directivity of the individual elements which compose the array. In the case of the proposed design, the assumption of uniform motion is not appropriate

since CMUT membranes, when excited, behave more like boundary-clamped plates. In addition, the motion profile of an excited element is affected by the presence of neighboring elements due to acoustic cross-talk. These effects are accounted for in the linear systems model by imposing an element factor correction based on the direction of the observation vector $\vec{r}$ with respect to the element normal. The correction is determined by comparing the element factor simulated in Field II (of a single element) with that simulated in the BEM model for an element surrounded by eight neighboring elements, all of which are also excited. The correction factor is shown in Fig. 47. Note that the correction is applied twice for each element, once based on the transmit direction, and a second time based on the receive direction.

5.4.2 Simulated tissue phantoms

Simulated tissue phantoms are critical for obtaining realistic estimates of SNR and penetration depth. To simulate tissue media, tissue attenuation and backscatter must be modeled. For a first-order analysis, we adopt a backscatter model based on the statistics of fully-realized speckle (Rayleigh) [133] and a simple model for path-dependent attenuation.

In this model, the backscatter strength of tissue is characterized by its backscattering coefficient (BSC), denoted by the symbol η_{bs}. The tissue is assumed to be composed of a large number of scattering sites, producing speckle patterns (the random interference produced by a set of wavefronts) with intensity that follows a Rayleigh probability density function. To simulate tissue, a dense collection of scatterers are placed randomly within a volume, and an additional convolution step representing the target response in (65) is added to the linear systems framework

$$b(t) = \frac{2\pi}{S_r}\sqrt{\frac{\eta_{bs}(t)}{n_s}} \tag{72}$$

where S_{rx} is the surface area of the receiving aperture, η_{bs} is the backscattering coefficient (BSC) of the tissue, and n_s is the scatterer density in terms of number of scatterers per unit volume. The BSC is defined as the backscattering cross-section per unit volume of insonified tissue, i.e.

$$\sigma_{bs} = \int_V \eta_{bs}\,\mathrm{d}V \tag{73}$$

The filter scales the backscattered response based on the frequency-dependent backscattering strength of the tissue. It is implented as a zero-phase finite impulse response (FIR) to ensure that no additional time delay is added to the response. To ensure that the speckle is fully-realized, a minimum

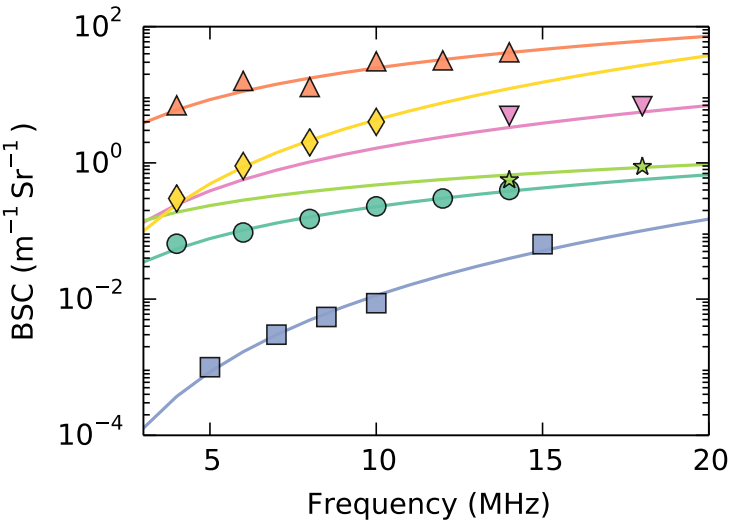


Figure 49: Backscattering coefficient spectra of different tissues measured experimentally by various authors (symbols) and their corresponding power-law fits (—). These curves can be used to design filters for simulation purposes. (Legend: calcified aortic wall (△) and normal aortic wall (◯) from Landini [8], subcutaneous fat (★) and dermis (▽) from Raju [9], canine myocardium (◊) from O'Donnell [10], and blood at 8% hematocrit (□) from Shung [11]).

Table 8: Tissue Attenuation

	α_{dB} (dB/cm–MHz^y)	y
Blood	0.14	1.21
Myocardium	0.52	1

Tissue properties obtained from [90, 134]

of 30 scatterers should be present in the focal volume, which corresponds to about 5000 scatterers per cubic centimeter.

Backscattering coefficient for a variety of tissue has been characterized experimentally in the literature [8–11]. These spectra and their corresponding power-law fits are shown in Fig. 49.

To simulate tissue attenuation, the final response is reduced based on the path through the phantom for each transmit element/scatterer and receive element/scatterer pair. The path length within each tissue section is determined algorithmically by first determining the intersection points with the tissue boundaries and then segmenting the total path (see Fig. 50). The attenuation constant for each tissue type is based on the transducer center frequency, meaning that frequency-dependent variation is ignored. The attenuation values used in simulation are shown in Table 8.

Two simulated phantoms are constructed for this study in order to emulate the cardiac environment. A penetration phantom (Fig. 51a) simulates a blood environment with 2 mm thick

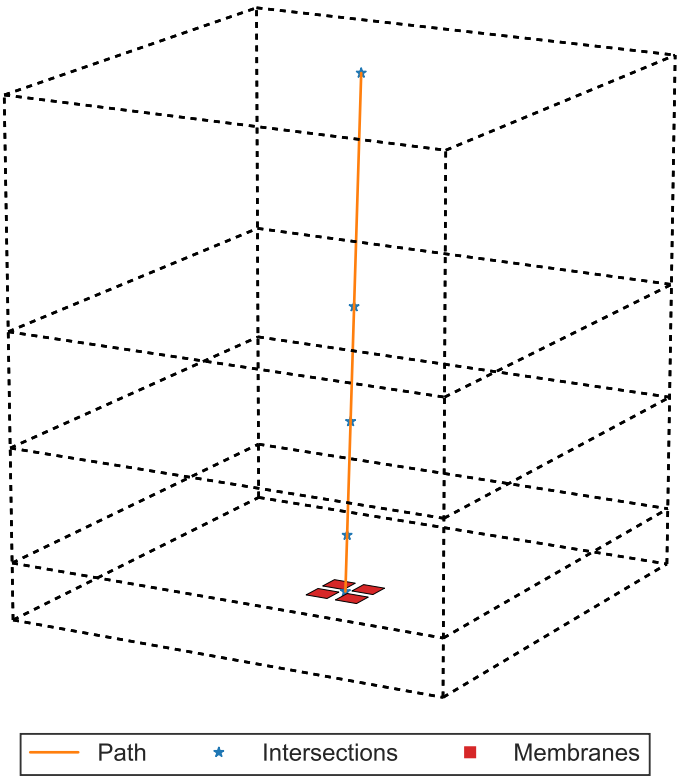
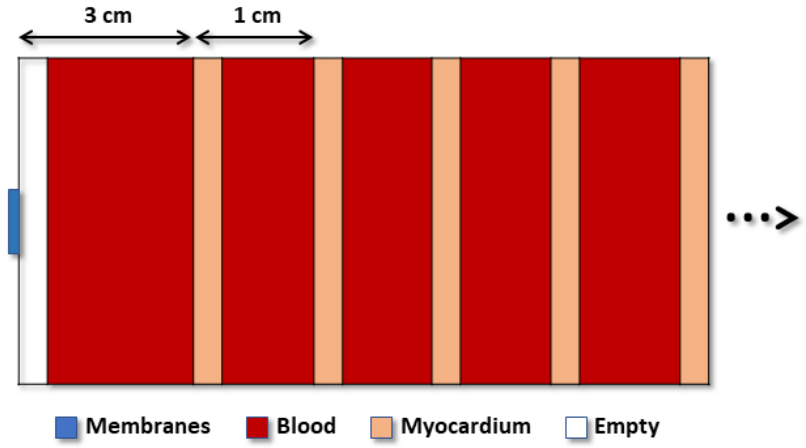


Figure 50: Schematic of a simulated tissue phantom with path-dependent attenuation.

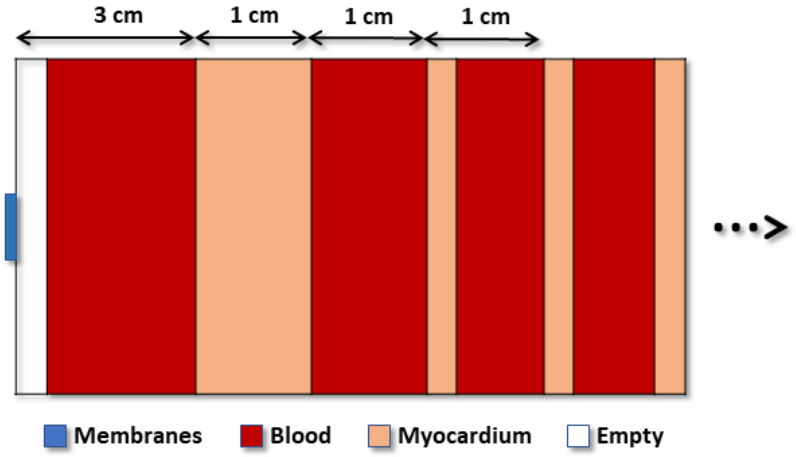
myocardium signal markers placed at 1 cm intervals, starting from 3 cm and stopping at 15 cm depth. A transmission phantom (Fig. 51b) simulates penetration beyond a 1 cm thick myocardium layer, again with 2 mm thick myocardium signal markers placed at 1 cm intervals. In both cases, a thin 1 mm spacer (empty) is placed in front of the array to avoid singularities in the spatial impulse responses. The phantom dimensions are 1 cm by 1 cm by 15.2 cm and contain around 75,000 scatterers.

Finally, rather than simulate the phantoms with the full array (which would be computationally prohibitive), the simulation is carried out for one transmit/receive element pair. The result is scaled up to the full array by considering the following argument. For an array of M transmit elements and focal distance $r \gg L$ where L is the aperture size, the summed transmitted pressure will add coherently, resulting in a received pressure with gain M. For an array of N receive elements, after beamformation and summation (averaging), the noise level will be reduced by a factor $1/\sqrt{N}$. An estimate for the SNR of the full array will therefore be

$$SNR_{array} \approx SNR_1 + 20log_{10}\left(M\sqrt{N}\right) \tag{74}$$



(a) Penetration phantom



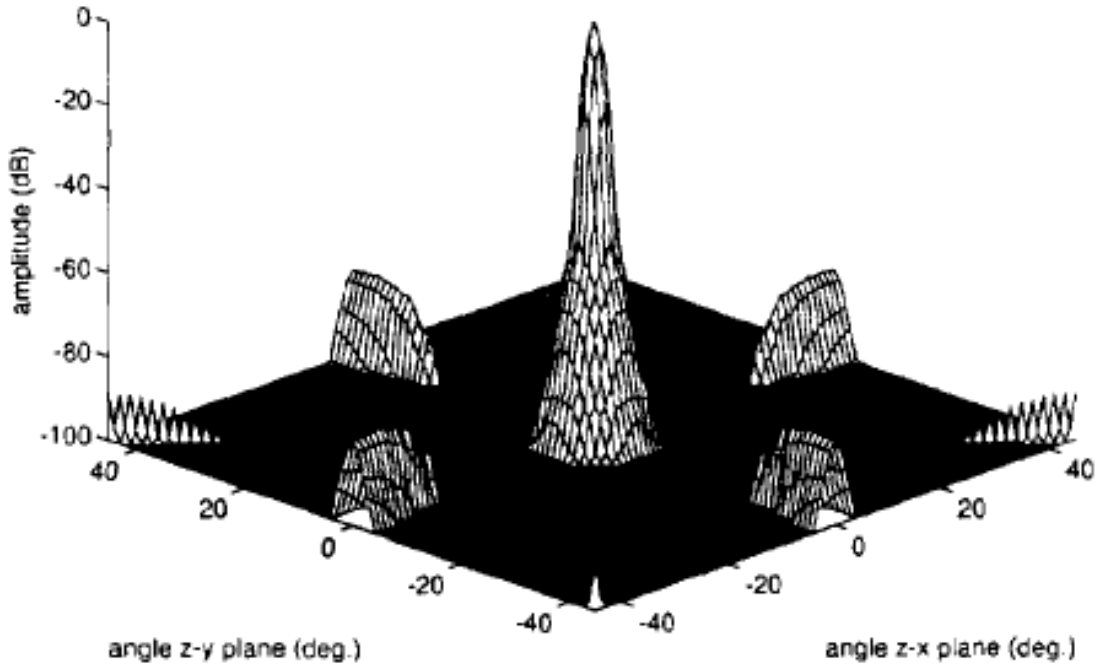
(b) Transmission phantom

Figure 51: Layer diagrams for the simulated tissue phantoms.

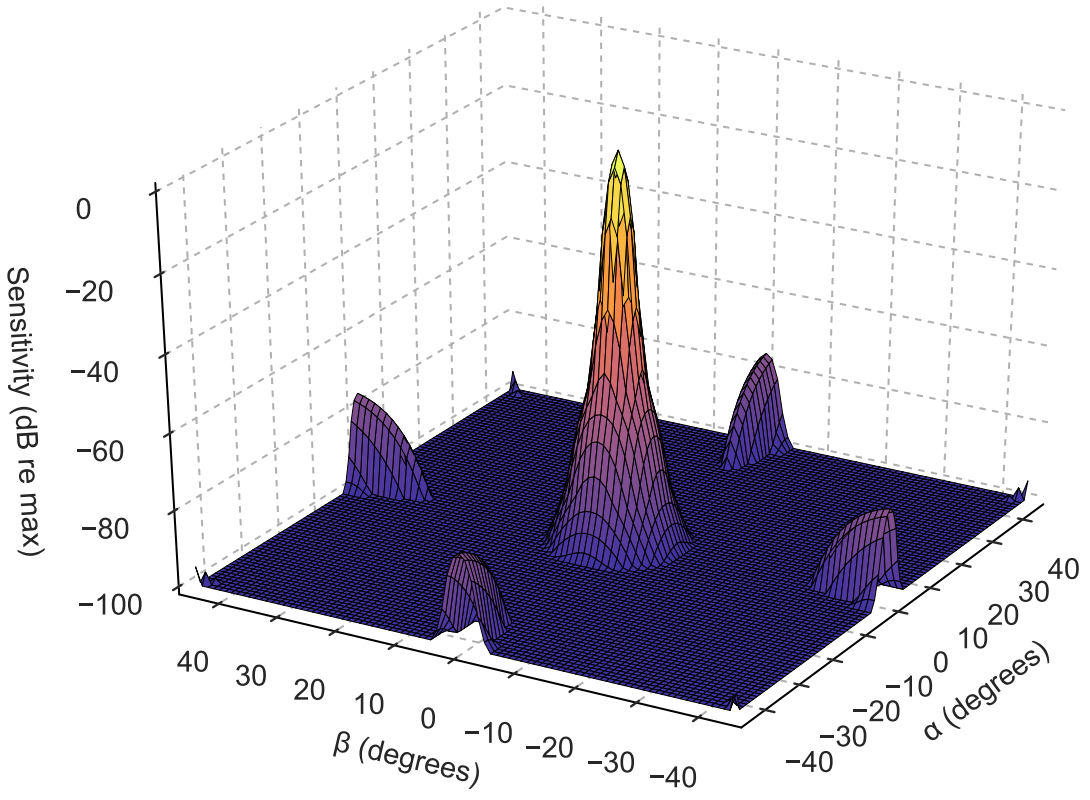
where SNR_1 is the SNR calculated for the single transmit/receive element pair.

5.4.3 Verification with Lockwood's original design

We verify our beamplot calculation methodology by simulating Lockwood's original design with 517 transmit and 517 receive elements [7]. In this design, the transmit and receive array pitches are selected with $p = 3$ and $d = (0.9)\lambda/2$ where λ is the wavelength corresponding to the transducer's center frequency. The excitation pulse is 7 MHz with 35% FBW. Beamplots are calculated for on-axis focus and steered focus at $(\alpha, \beta) = (40°, 0)$ at a distance corresponding to f/4. Note that Lockwood et. al., using analytical formulations for rectangular apertures, generated full-image PSFs over many scanlines which were then collapsed to two dimensions by considering the *worst-case* levels along each scanline.

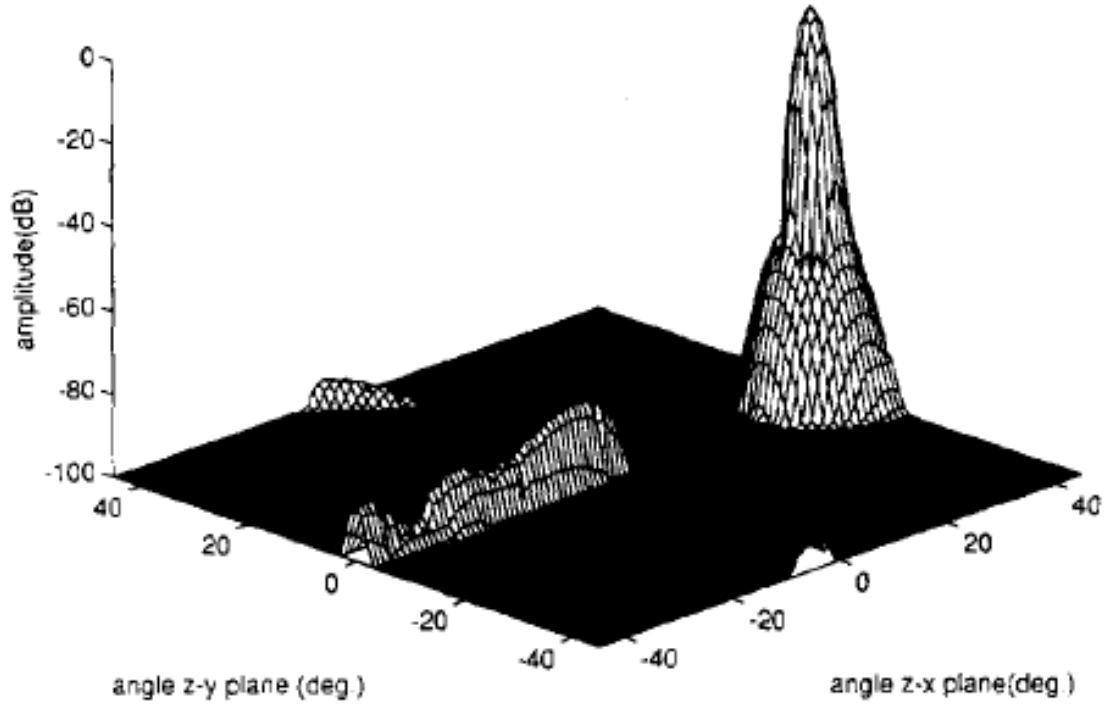


(a) On-axis image PSF generated by Lockwood. Copyright 1994 IEEE. Reprinted, with permission, from [7].

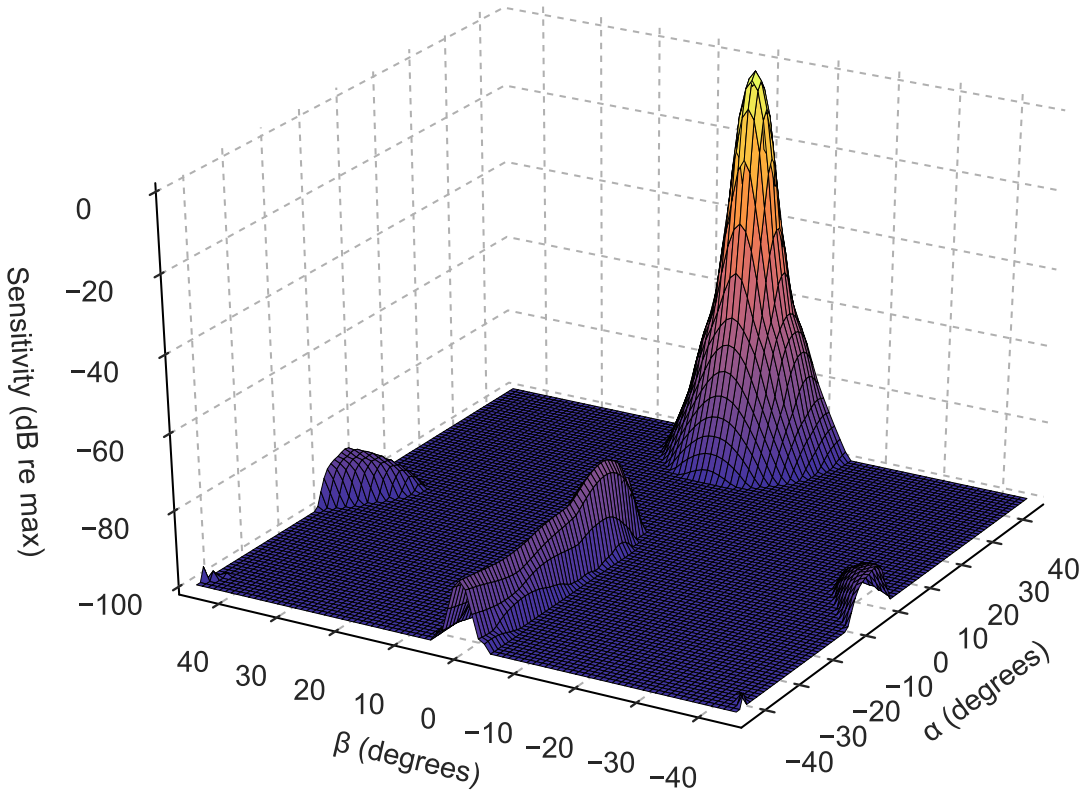


(b) On-axis beamplot generated using simulation methodology.

Figure 52: Comparison of on-axis image PSF and two-way beamplot for Lockwood's original sparse vernier array design.



(a) Steered image PSF generated by Lockwood. Copyright 1994 IEEE. Reprinted, with permission, from [7].



(b) Steered beamplot generated using simulation methodology.

Figure 53: Comparison of steered image PSF and two-way beamplot for Lockwood's original sparse vernier array design.

A comparison of the beamplots generated is shown in Fig. 52 for on-axis focus, and in Fig. 53 for steered focus. The agreement with Lockwood's results is very close, validating our simulation methodology and confirming that the two-way beamplot is an excellent predictor of PSF characteristics.

5.5 Imaging performance results

5.5.1 Beamplots

The simulation methodology was applied to assess the imaging performance of the proposed Vernier array design. Beamplots were generated for all combinations of apodization (on/off), element factor (on/off), and focus position (on-axis $(0^\circ, 0^\circ)$, steered $(40^\circ, 0^\circ)$) in order to isolate the effects of each parameter. In all cases the depth of focus was set to 5 cm.

A comparison of the on-axis beamplots with and without apodization is shown in Fig. 54 (surface plots) and Fig. 55 (contour plots). As expected, apodization results in an increase in main-lobe width and a significant smoothing in side-lobe structure. However, apodization is shown to not reduce side-lobe levels significantly. This can be explained by the large fractional bandwidth of the pulse (80%) [90] which results in smoothing of the side-lobe structure from superposition of a wide-range of frequencies. The steered beamplots with and without apodization show a similar result. When steered, apodization is found to increase the grating lobe level by 3 dB.

The addition of element factor is expected to help in side-lobe suppression when focusing on-axis by decreasing the sensitivity of the array to off-axis targets. This can be understood analytically by the product theorem, which states that, in the far-field, the directivity of an array with element factor is the directivity of the array composed of simple sources modulated by the element factor directivity [78, 90]. The on-axis beamplots with and without element factor are shown in Fig. 58 (surface plots) and Fig. 59 (contour plots). Since the element factor is applied twice for pulse-echo, once in transmit and once in receive, the side-lobe suppression is significant–a reduction as much as 10.6 dB.

When the beam is steered, the modulation due to element factor will reduce the peak sensitivity of the main-lobe while boosting side-lobe levels which now appear on-axis. This effect is observed in the steered beamplots shown in Fig. 60 (surface plots) and Fig. 61 (contour plots).

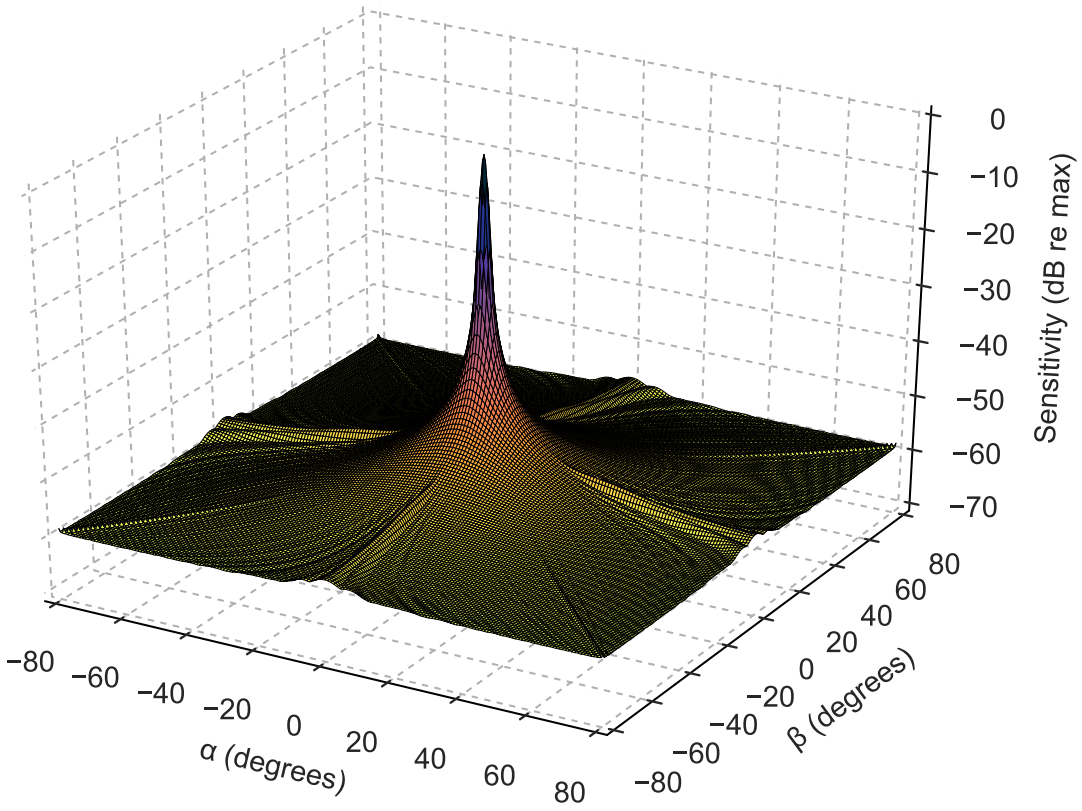
Finally, with both apodization and element factor applied, the resulting on-axis beamplot (Fig. 62) and steered beamplot (Fig. 63) have the combined smoothing and modulating effects.

To quantify the observed changes in the beamplot, -6 dB resolution and CTR for each combination is tabulated in Table 9. Apodization is observed to be detrimental to both resolution (+0.5 mm) and CTR (+2-4 dB). The increase in CTR is attributed predominantly to the widening of the main-lobe. Element factor is observed to have no effect on resolution, to improve the on-axis CTR slightly (-0.7 dB), and to degrade the steered CTR slighlty (+0.04 dB). Although element factor appears

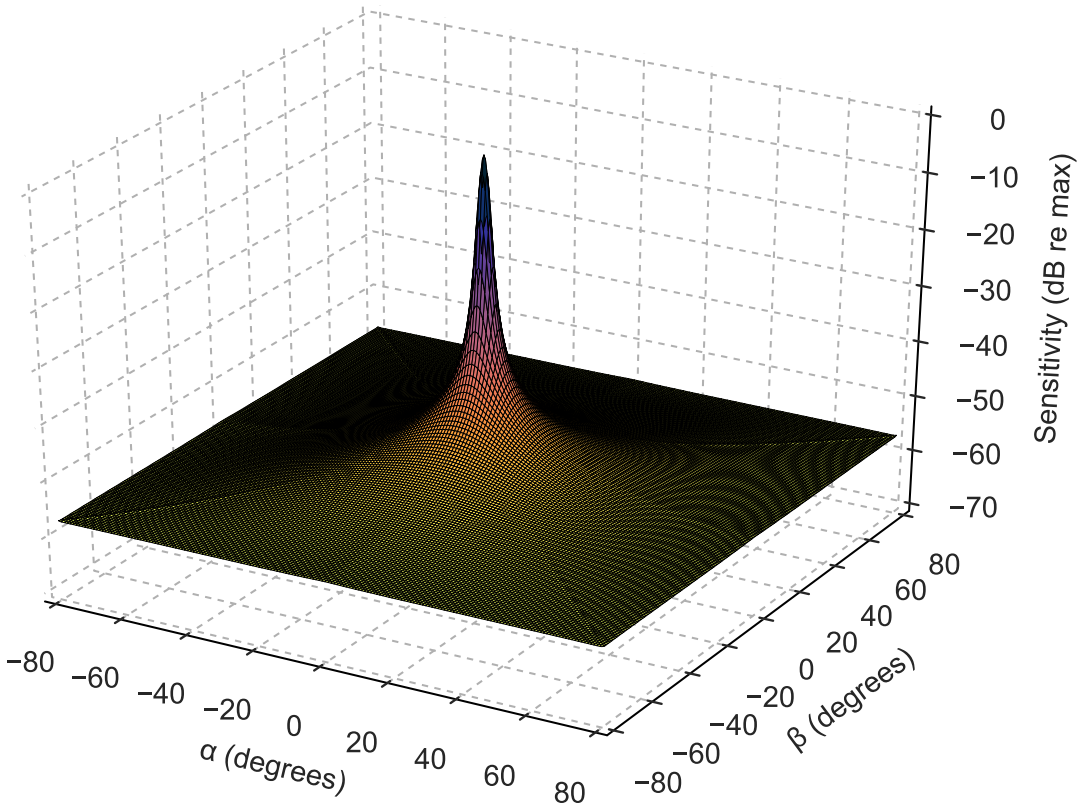
Table 9: Resolution and CTR

Focus (5 cm)	Apodization	Element Factor	Resolution (mm)	CTR (dB)
$(0°, 0°)$	No	No	1.7	-9.52
$(0°, 0°)$	Yes	No	2.3	-6.88
$(0°, 0°)$	No	Yes	1.7	-10.23
$(0°, 0°)$	Yes	Yes	2.3	-7.22
$(40°, 0°)$	No	No	2.4	-6.27
$(40°, 0°)$	Yes	No	3.1	-4.18
$(40°, 0°)$	No	Yes	2.4	-6.23
$(40°, 0°)$	Yes	Yes	3.1	-4.15

to have a significant effect on beamplot structure, these effects are not manifested in the resolution (main-lobe structure is unaffected) and CTR (changes in structure occur at very low levels).

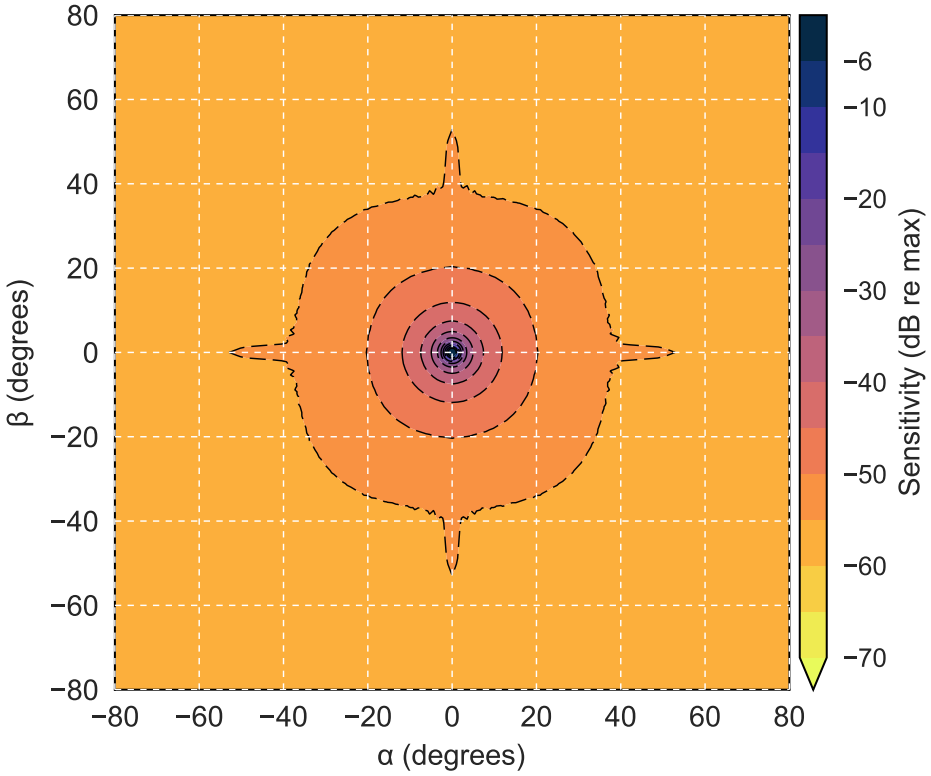


(a) On-axis beamplot surface without apodization (base case)

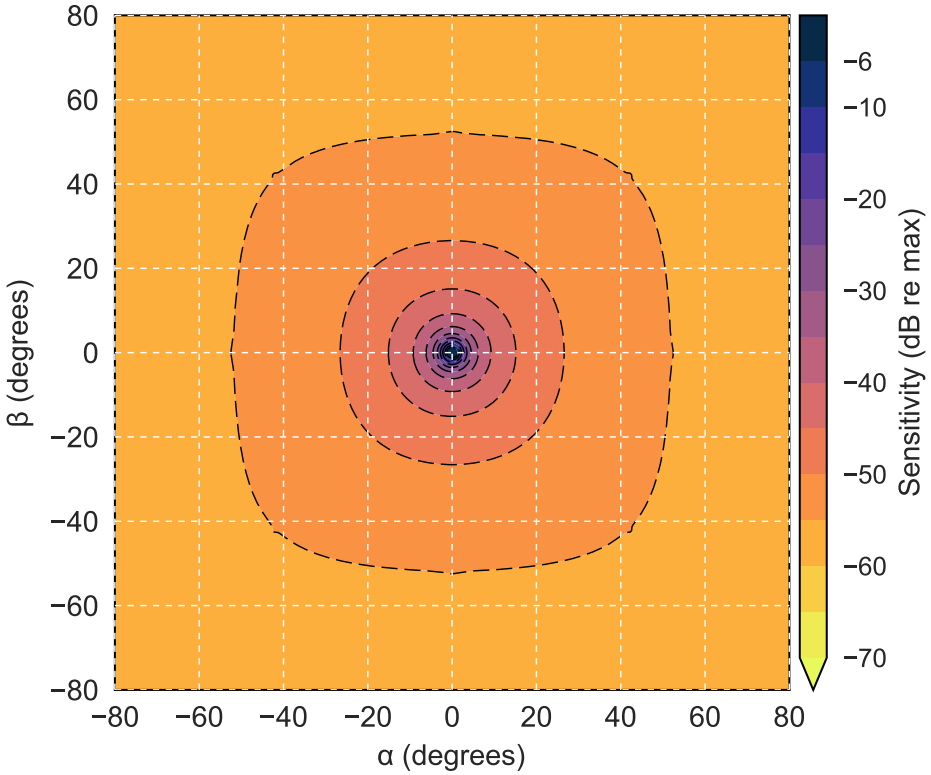


(b) On-axis beamplot surface with apodization

Figure 54: Comparison of on-axis beamplot surface with and without apodization (element factor not applied).

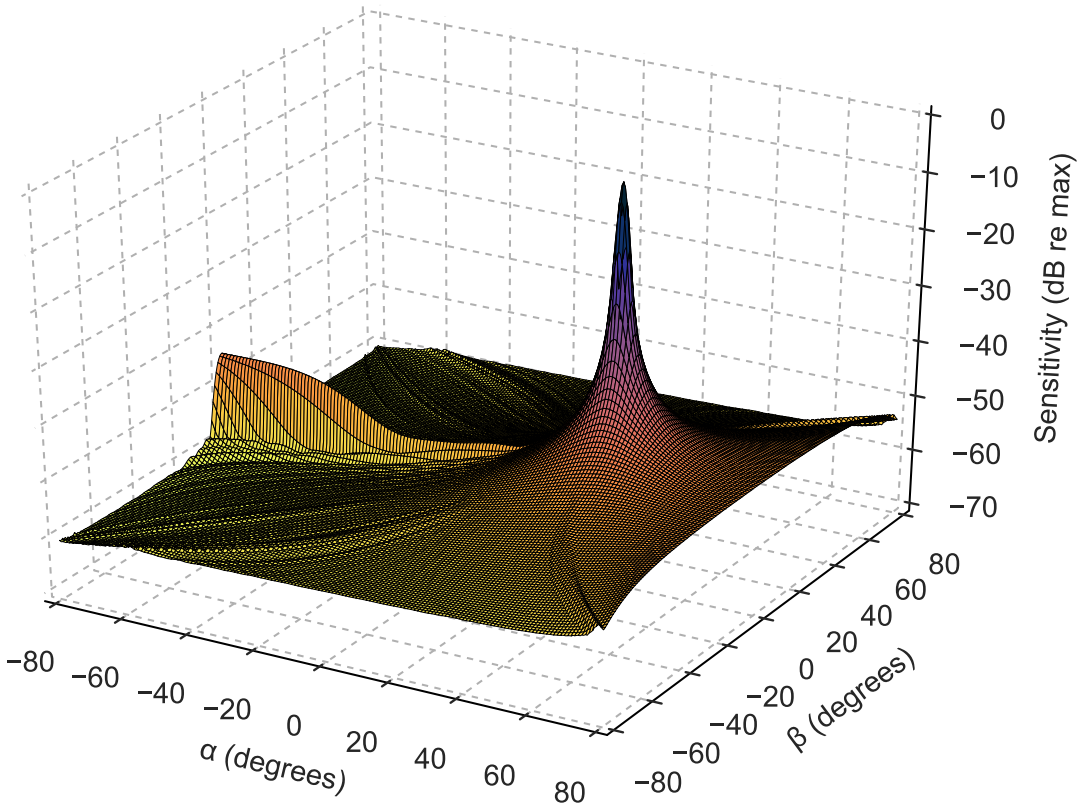


(a) On-axis beamplot contour without apodization (base case)

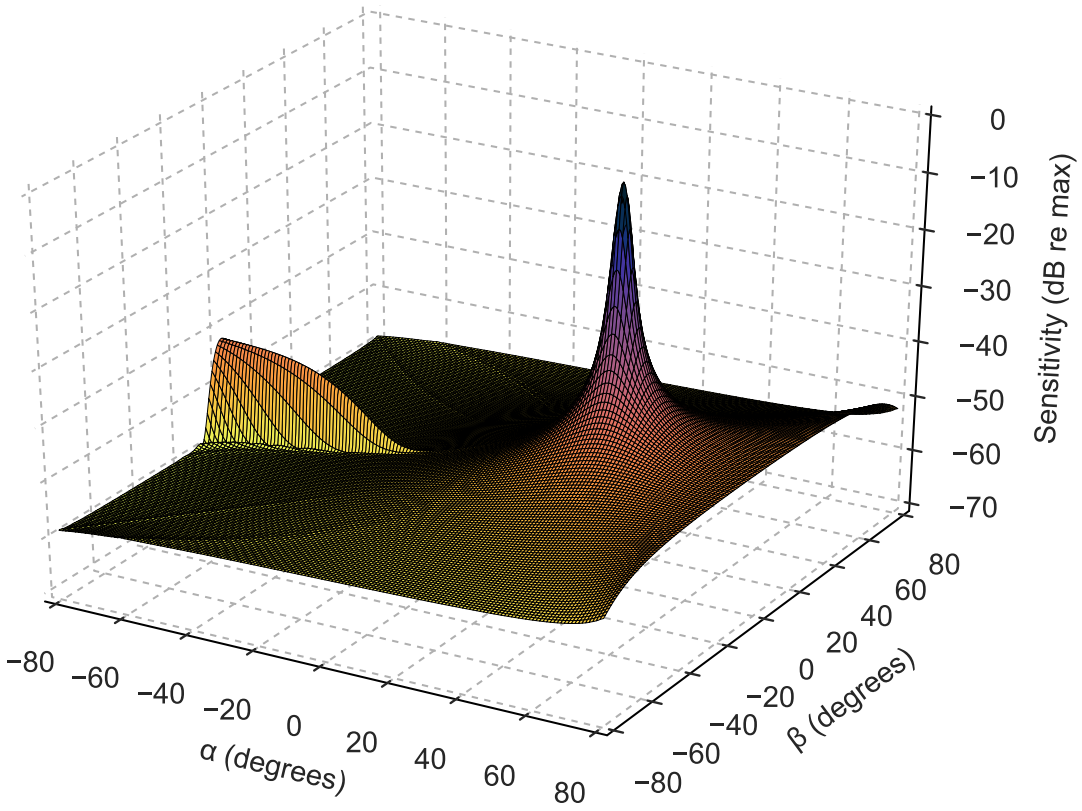


(b) On-axis beamplot contour with apodization

Figure 55: Comparison of on-axis beamplot contour with and without apodization (element factor not applied).

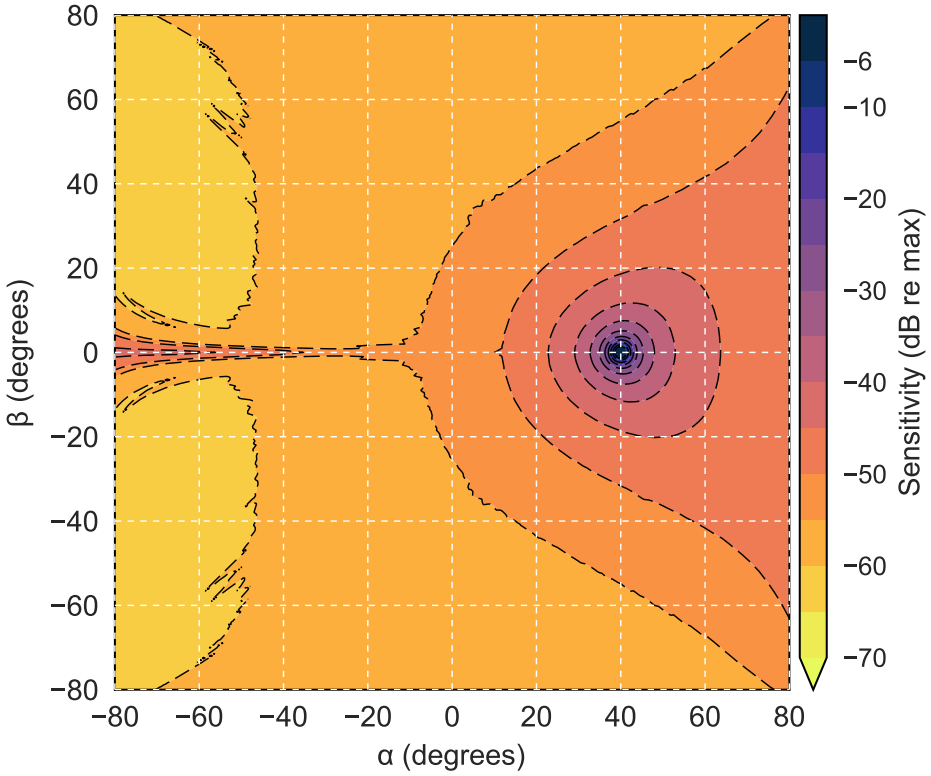


(a) Steered beamplot surface without apodization (base case)

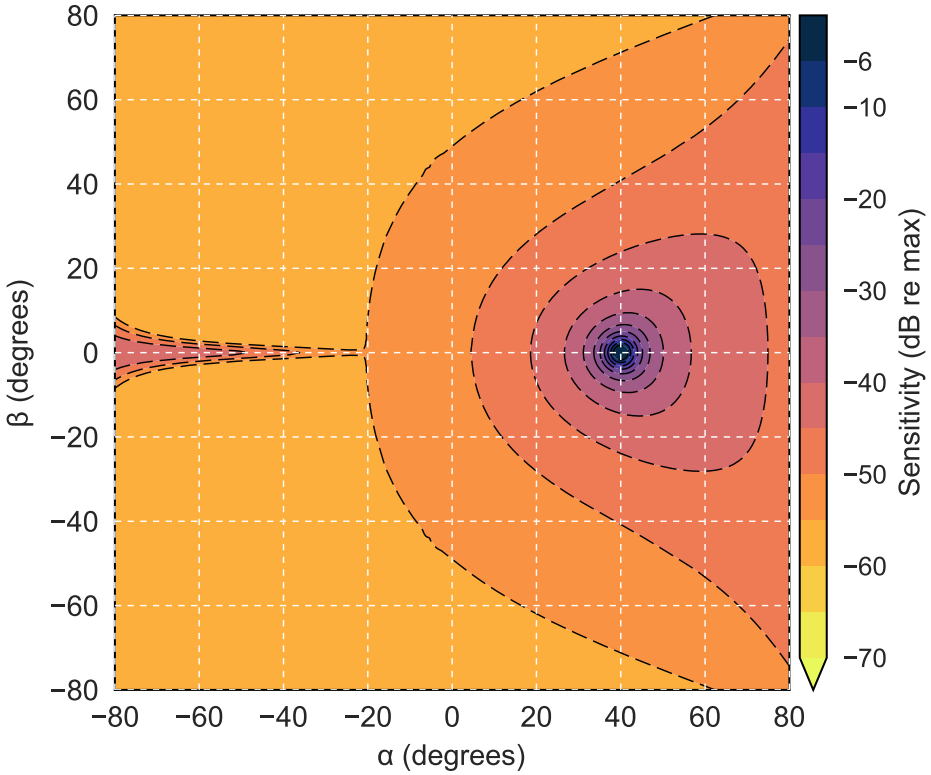


(b) Steered beamplot surface with apodization

Figure 56: Comparison of steered beamplot surface with and without apodization (element factor not applied).

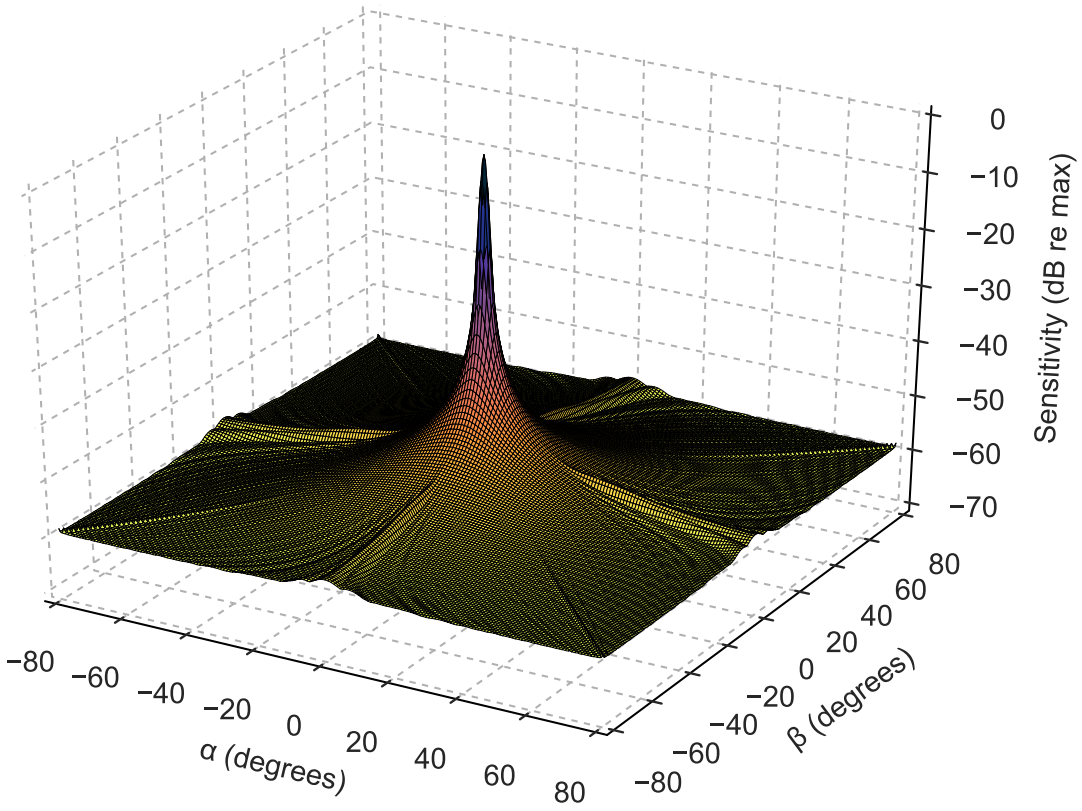


(a) Steered beamplot contours without apodization (base case)

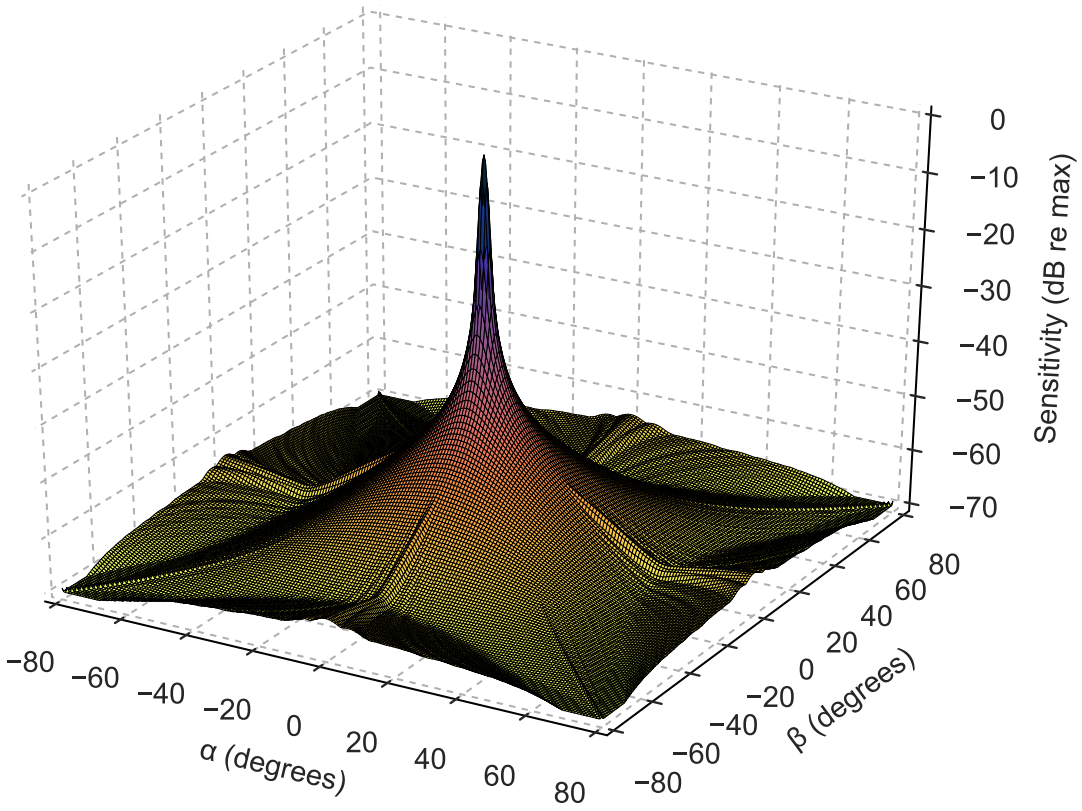


(b) Steered beamplot contours with apodization

Figure 57: Comparison of steered beamplot contours with and without apodization (element factor not applied).

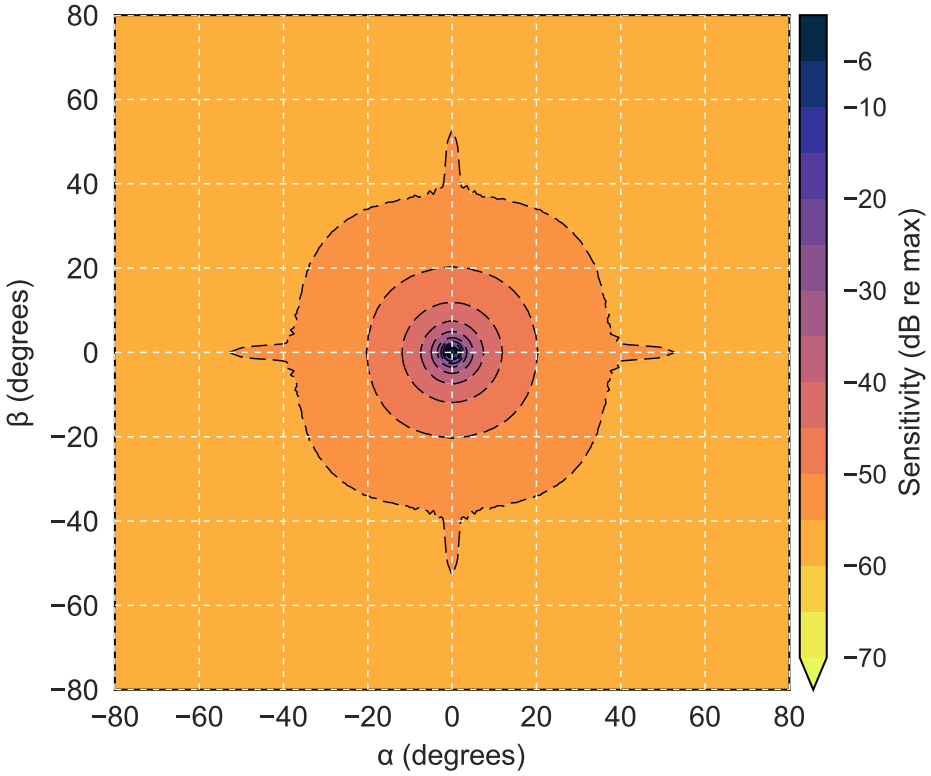


(a) On-axis beamplot surface without element factor (base case)

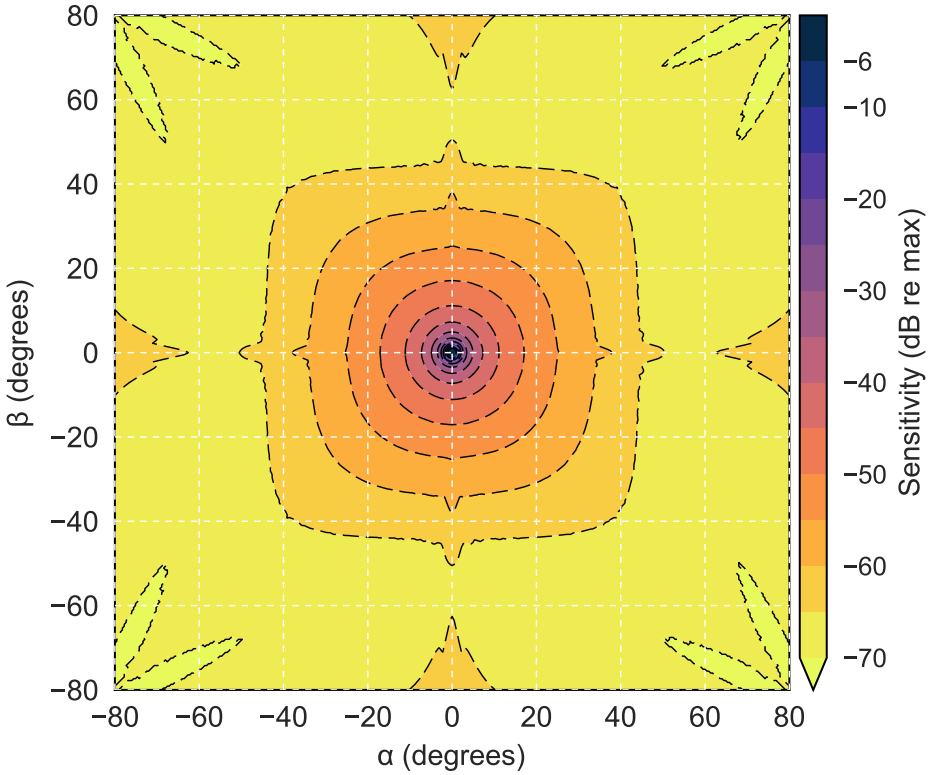


(b) On-axis beamplot surface with element factor

Figure 58: Comparison of on-axis beamplot surface with and without element factor (apodization not applied).

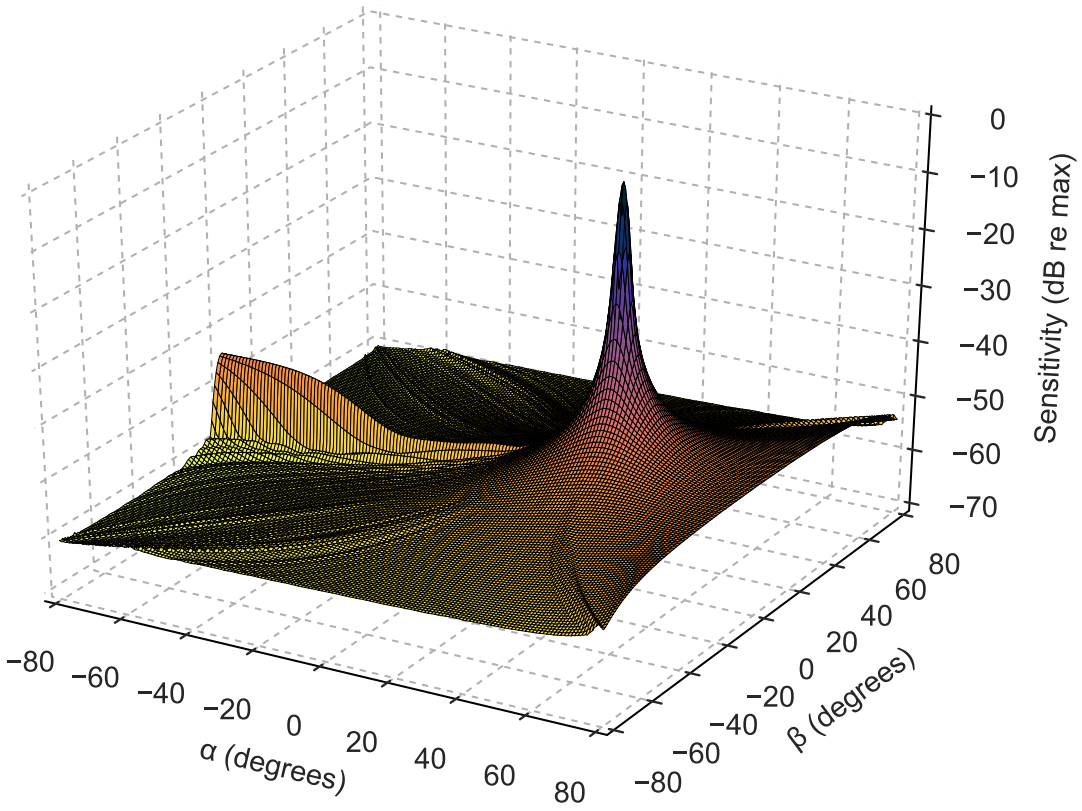


(a) On-axis beamplot contours without element factor (base case)

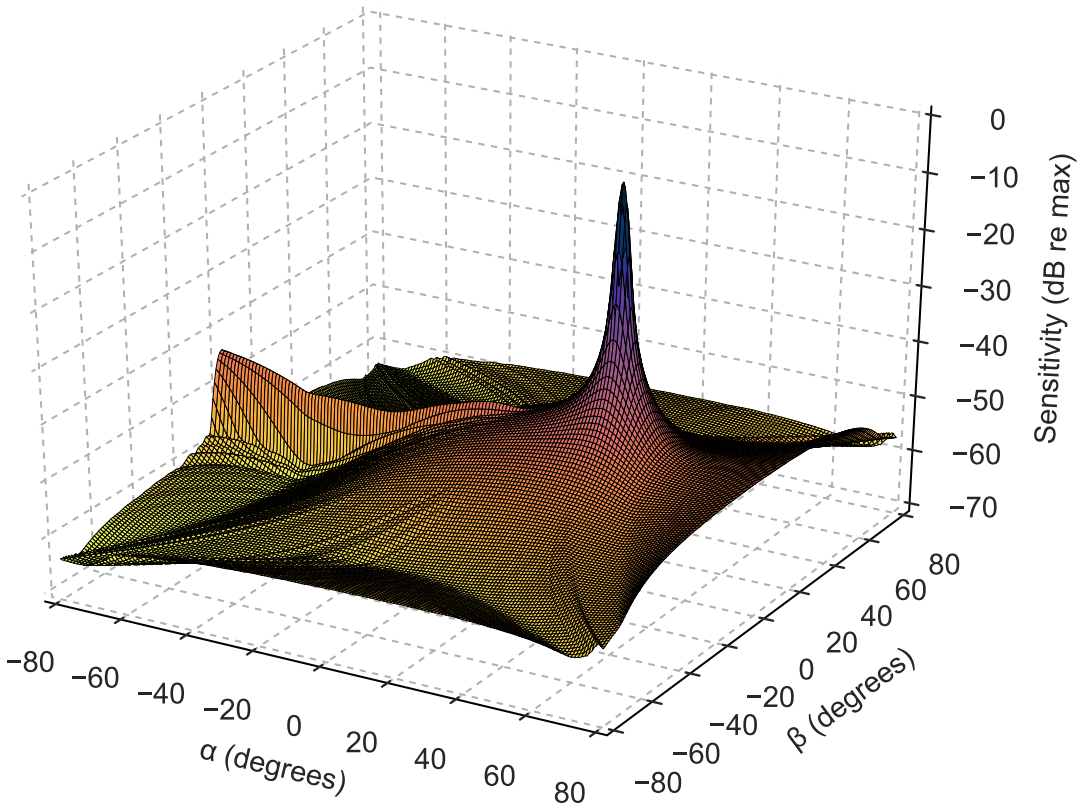


(b) On-axis beamplot contours without element factor

Figure 59: Comparison of on-axis beamplot contours with and without element factor (apodization not applied).

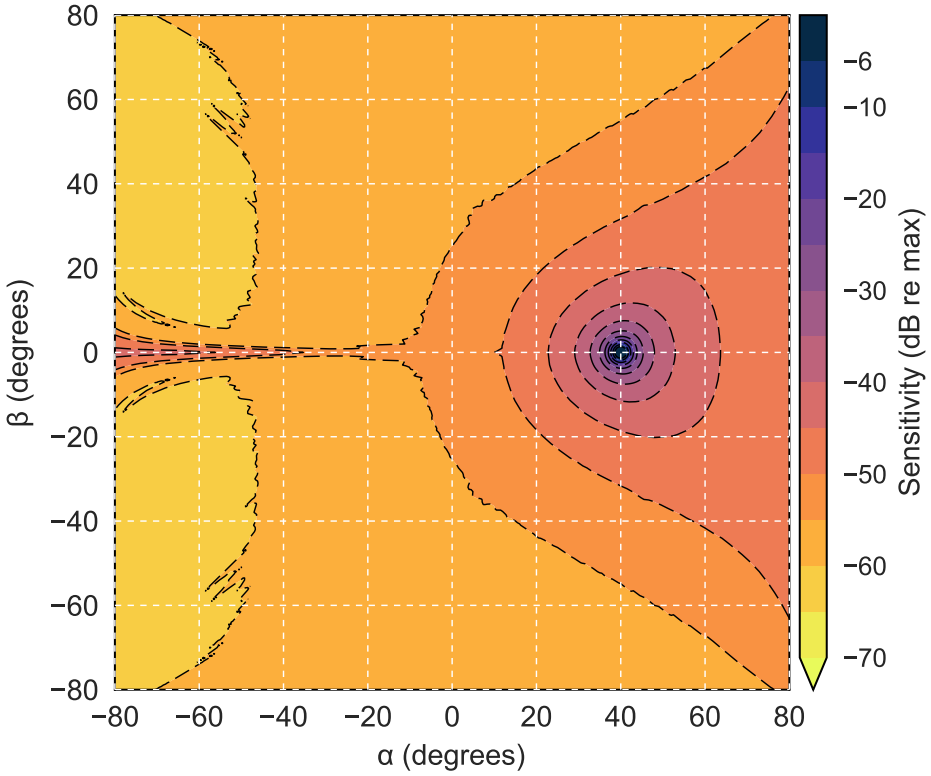


(a) Steered beamplot surface without element factor (base case)

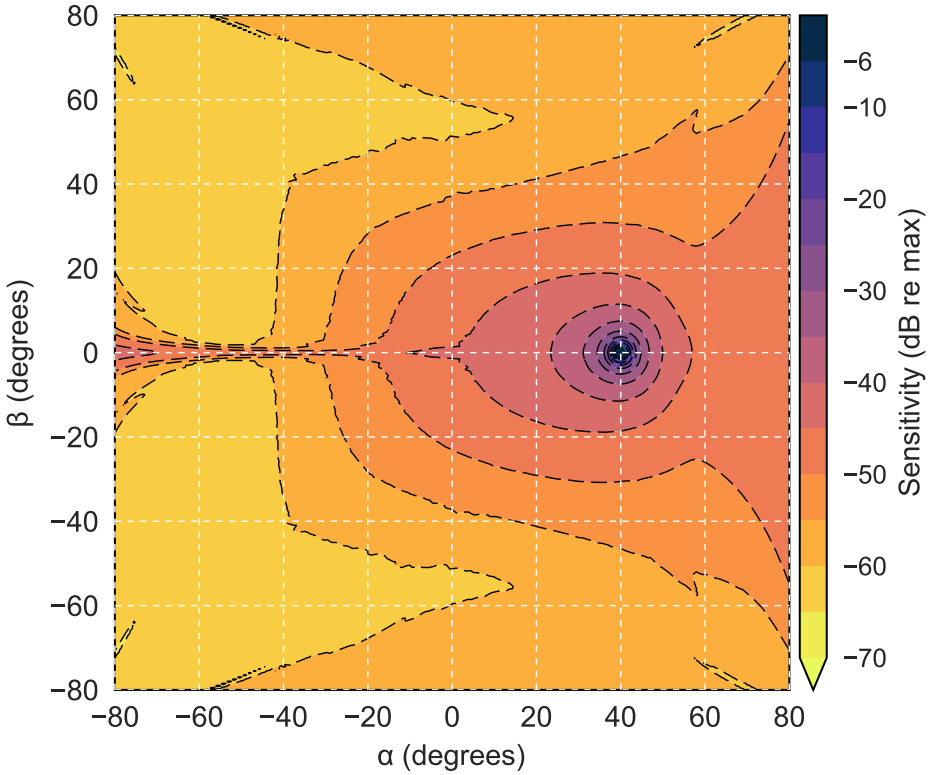


(b) Steered beamplot surface with element factor

Figure 60: Comparison of steered beamplot surface with and without element factor (apodization not applied).

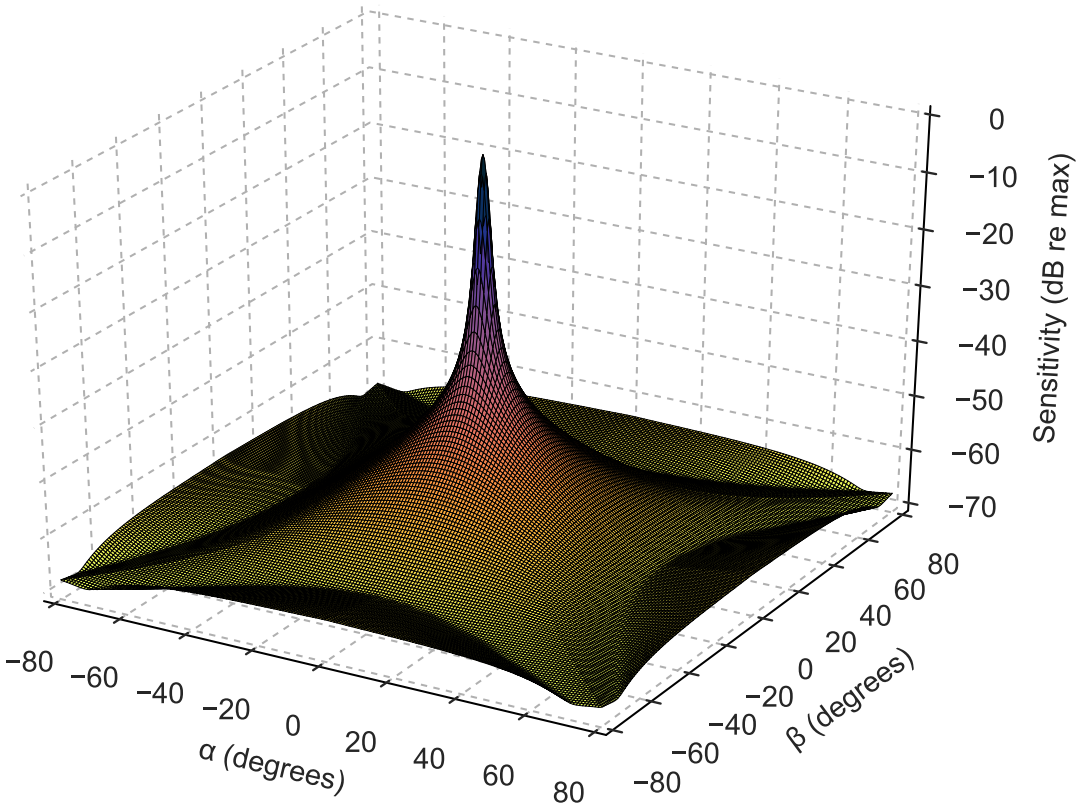


(a) Steered beamplot contours without element factor (base case)

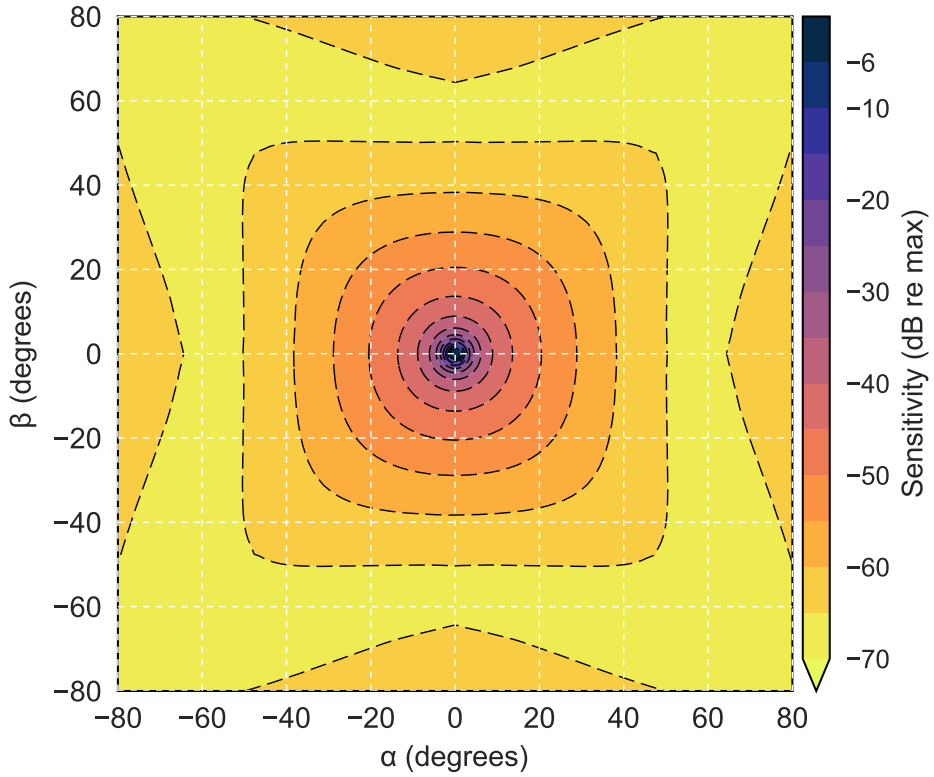


(b) Steered beamplot contours with element factor

Figure 61: Comparison of steered beamplot contours with and without element factor (apodization not applied).

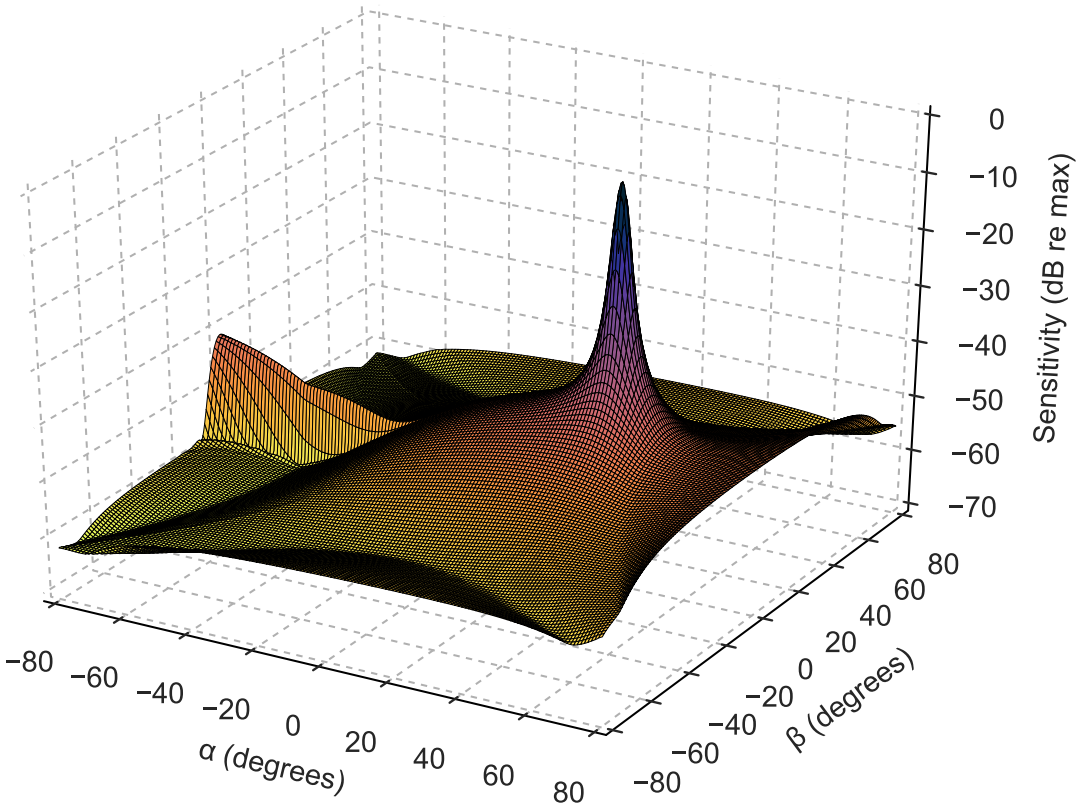


(a) On-axis beamplot surface

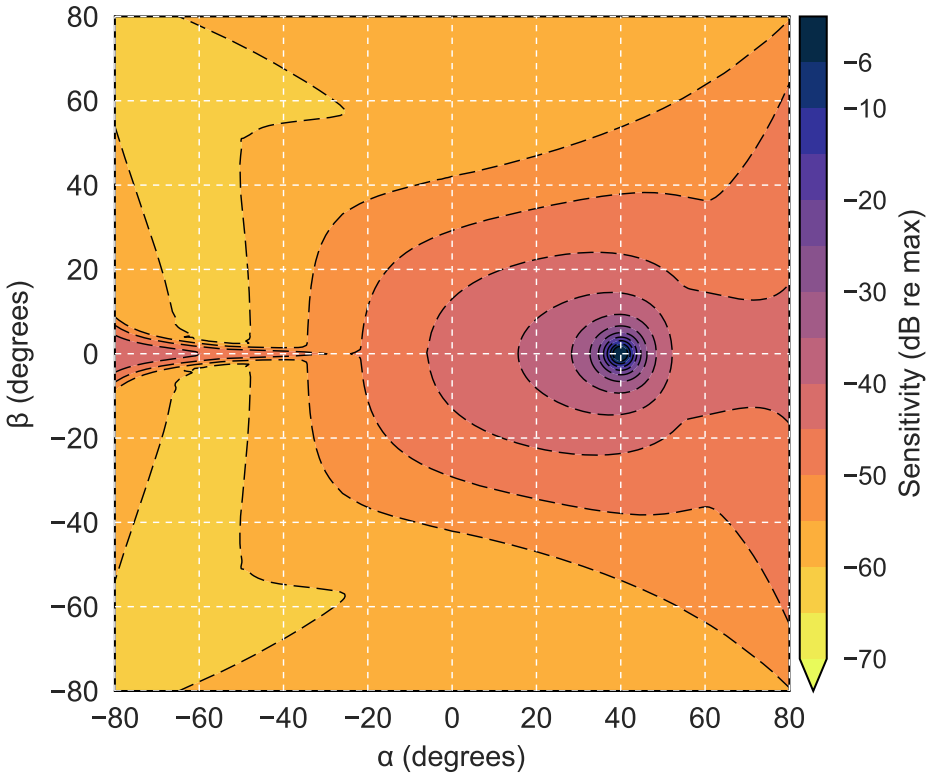


(b) On-axis beamplot contours

Figure 62: On-axis beamplot with both apodization and element factor.

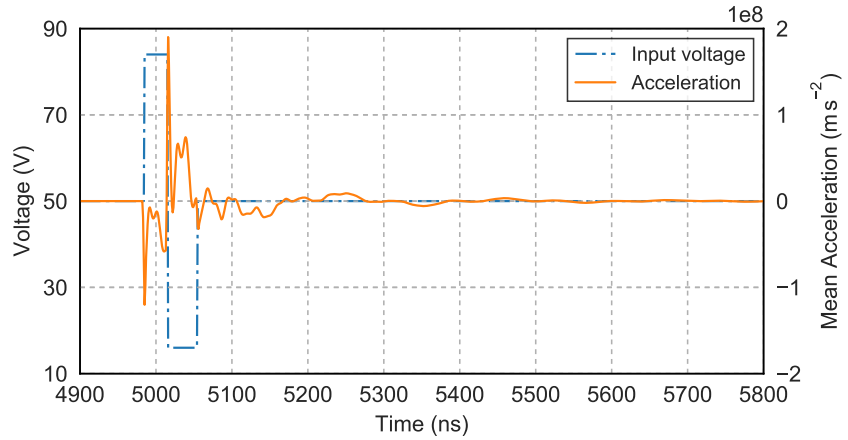


(a) Steered beamplot surface

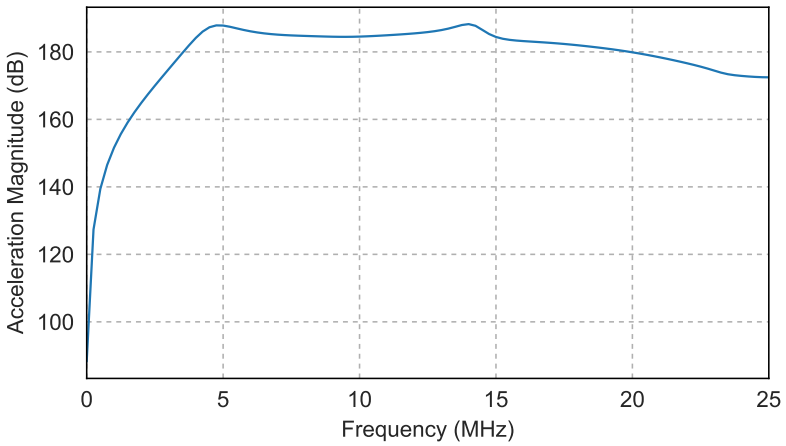


(b) Steered beamplot contours

Figure 63: Steered beamplot with both apodization and element factor.



(a) Input voltage pulse and the resulting acceleration response



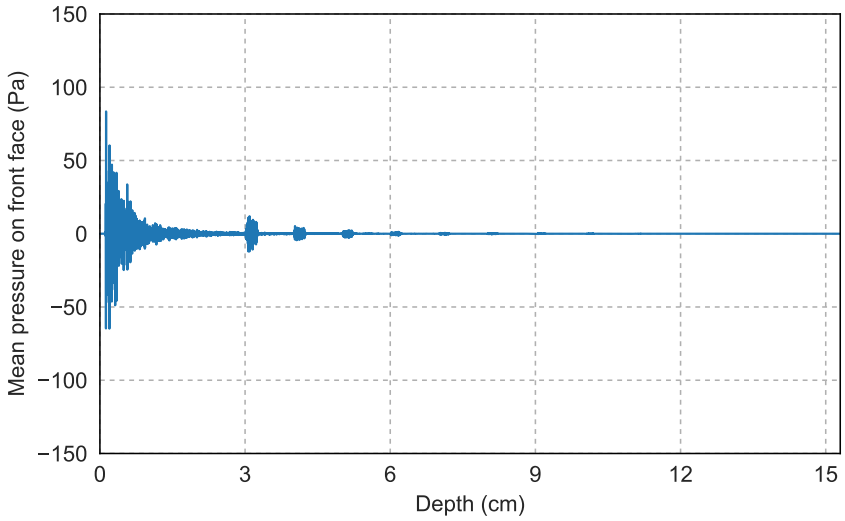
(b) Acceleration spectrum

Figure 64: Mean (normal) acceleration response calculated using a CMUT large-signal transient model.

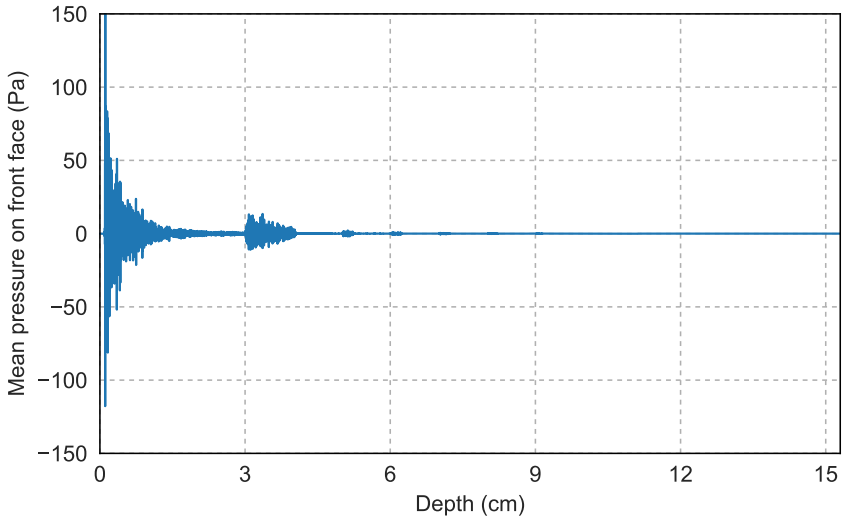
5.5.2 Transmission and penetration depth

SNR was assessed using the simulated transmission and penetration phantoms described previously. In order to generate the largest pressure possible from the array, the CMUT membranes should be driven through their full-swing to maximize volumetric displacement. However, the behavior of the membranes is complicated by the fact that, for large driving voltages, the electrostatic force has a non-linear dependence on the voltage squared over the gap squared. To account for this important non-linear effect, the single element acceleration response was generated using a large-signal transient model [75]. The input voltage (50 V DC bias with 34 V peak-to-peak square bipolar pulse) and resulting acceleration response are shown in Fig. 64.

The mean backscattered RF pressures from the phantoms are plotted in Fig. 65. The responses after envelope-detection and log compression are showin in Fig. 66. The minimum detectable



(a) Penetration phantom



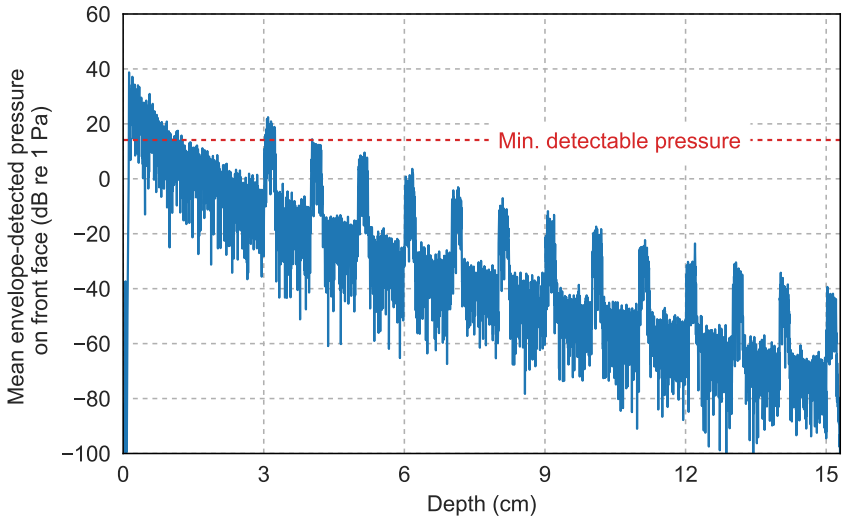
(b) Transmission phantom

Figure 65: Mean backscattered pressure from the simulated phantoms.

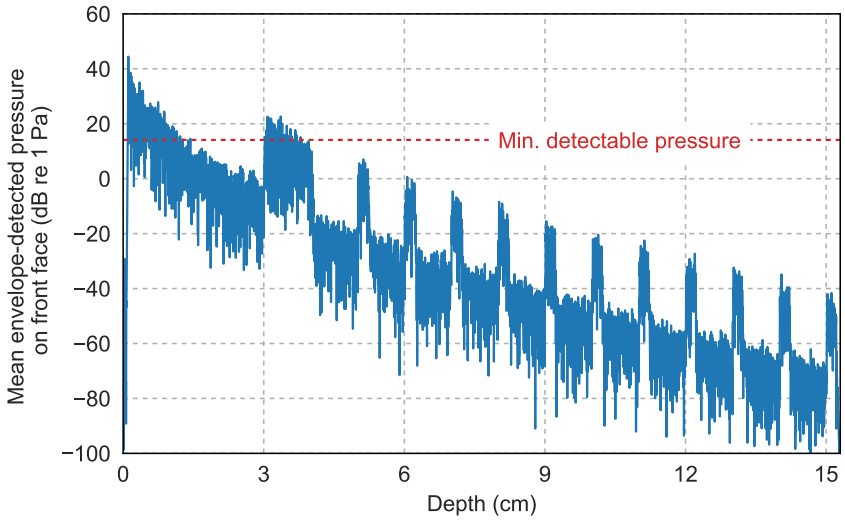
pressure, calculated from (63), was determined to be about 5 Pa (14 dB re 1 Pa). At this level, a single element is predicted to be able to detect myocardial backscatter from a 3–4 cm depth.

When the backscattered pressure is adjusted using (75) for 517 transmit and 517 receive elements, the predicted gain is 81 dB. The penetration depth (see Fig. 67) is improved to 15 cm and most of the reconstructed image is predicted to be dominated by speckle noise from blood. The transmission phantom also portrays a similar result, indicating that transmission through a 1 cm heart wall will not pose a problem.

To visualize these results more clearly, the mean SNR from each layer is plotted as a function of the depth of each layer (specifically, its midpoint) in Fig. 68. This data is fitted to a simple two-way



(a) Penetration phantom



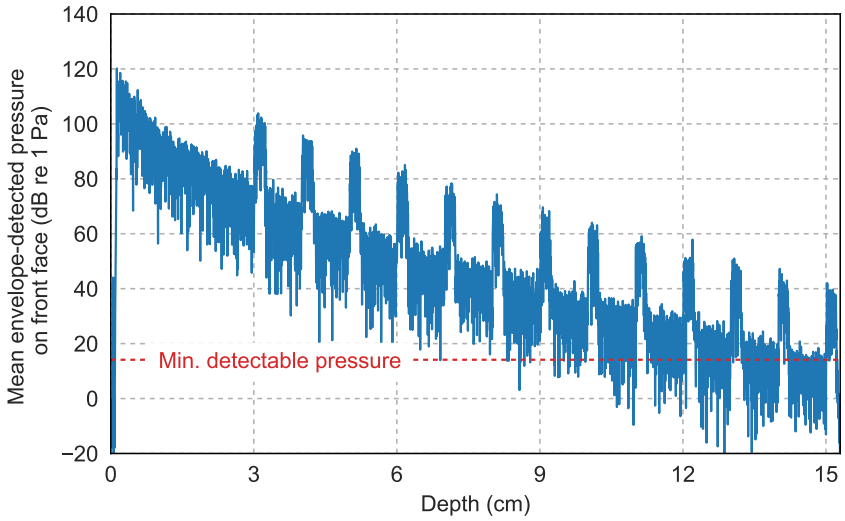
(b) Transmission phantom

Figure 66: Mean envelope-detected pressure from the simulated phantoms.

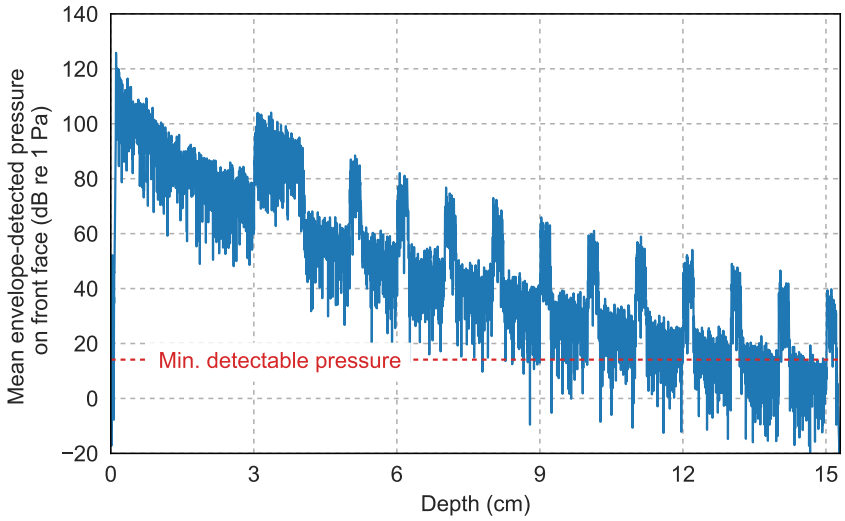
backscatter model with attenuation and spherical spreading, i.e.

$$p(r) = \frac{A}{r^2} e^{2Br} \tag{75}$$

for fit parameters A and B. The SNR drops below 0 dB at a depth of 13.5 cm for blood and at a depth greater than 15 cm for myocardium.

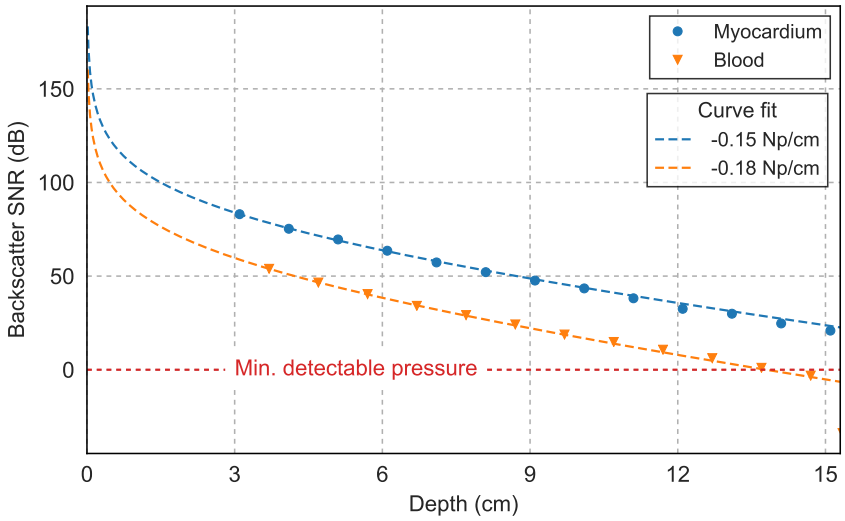


(a) Penetration phantom

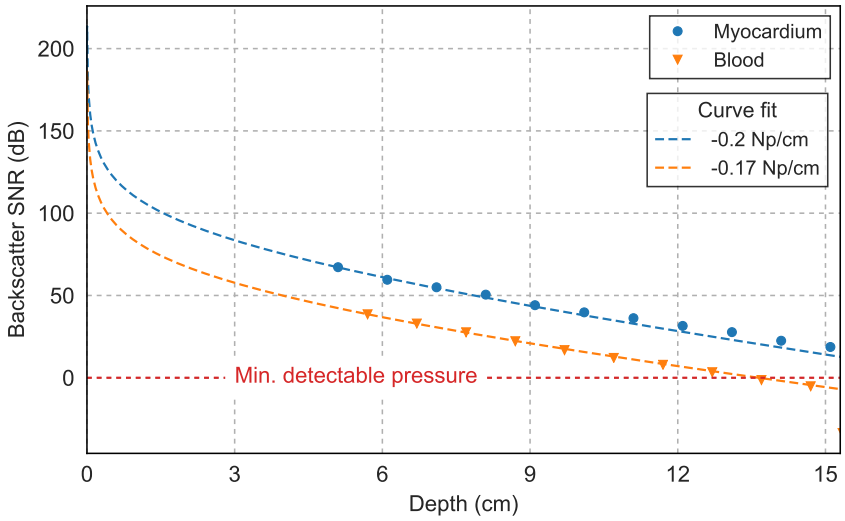


(b) Transmission phantom

Figure 67: Mean envelope-detected pressure from the simulated phantoms with adjustment for array gain.

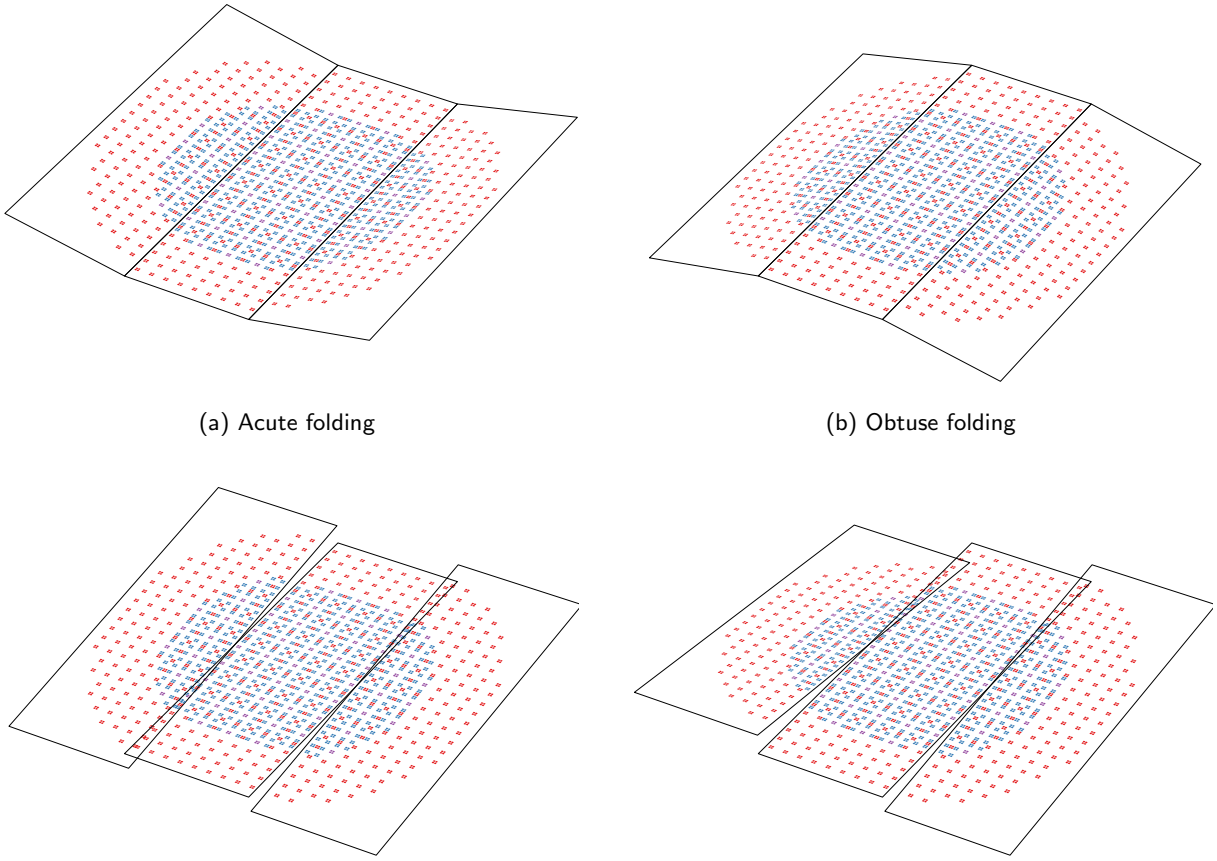


(a) Penetration phantom



(b) Transmission phantom

Figure 68: Backscattered SNR from each layer of the simulated phantoms with adjustment for array gain. The data is fitted to a simple two-way backscatter model with attenuation and spherical spreading.



(a) Acute folding

(b) Obtuse folding

(c) Aligned twist

(d) Opposed twist

Figure 69: Different types of orientation errors.

5.6 Performance degradation due to orientation errors

An anticipated complication of a foldable array design is the potential for incomplete knowledge of the element spatial locations. This could be due to the panes not deploying completely and perhaps from mechanical flex within the hinge mechanism itself. Because beamformation relies on precise information of the relative element locations, positional errors may result in a degraded beamplot and an overall decrease in imaging performance. To study this problem in more detail, we simulated performance degradation of the on-axis beamplot due to two types of orientation errors. *Folding* errors, which is the primary area of concern, refers to the angle of mismatch between a completely flat pane and a pane rotated about the hinge axis. Also distinguished are two folding error subtypes: acute folding errors, which occur when the pane fails to deploy completely, and obtuse folding errors, which occur when the pane deploys beyond the stop point. *Twist* errors, which may result from mechanical flex of the hinge, refers to the angle of mismatch between a completely flat pane and a pane rotated perpendicular to the hinge axis. Again, two subtypes can

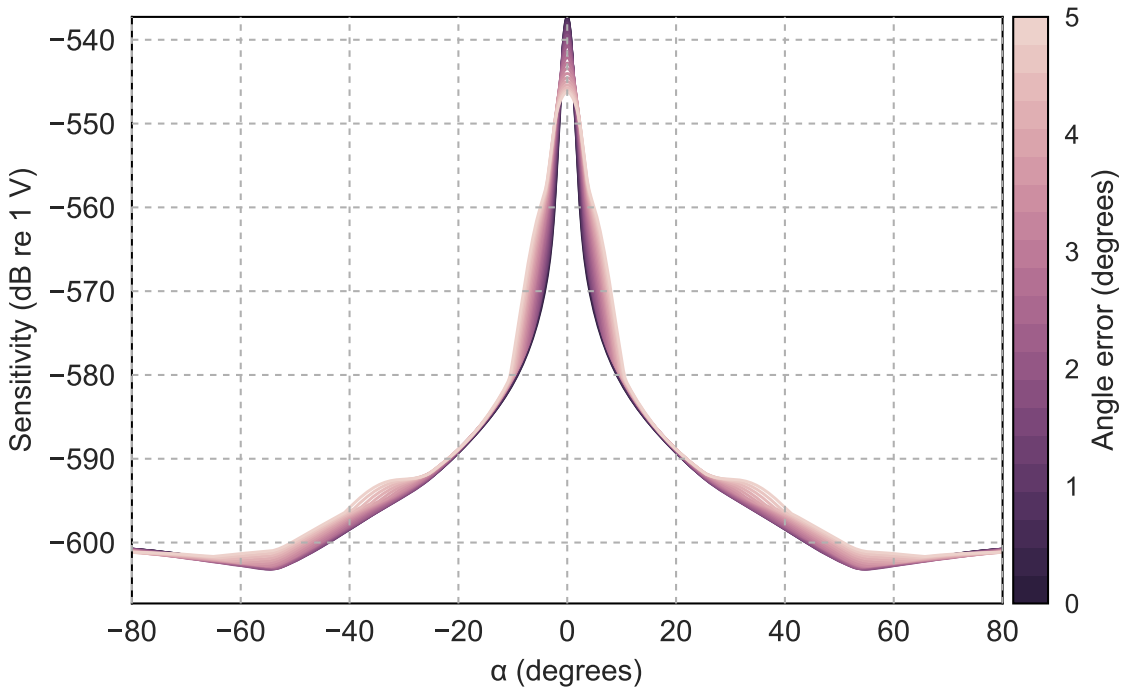
be identified: one in which the panes twist in the same direction (*aligned*), and one in which the panes twist in opposite directions (*opposed*). These orientation errors are shown in Fig. 69.

For both folding and twist errors, beamforming was performed assuming the array is completely flat. The angle of the error was swept from -5° to 5°. We considered slices of the beamplot along the z-x (α) and z-y (β) planes.

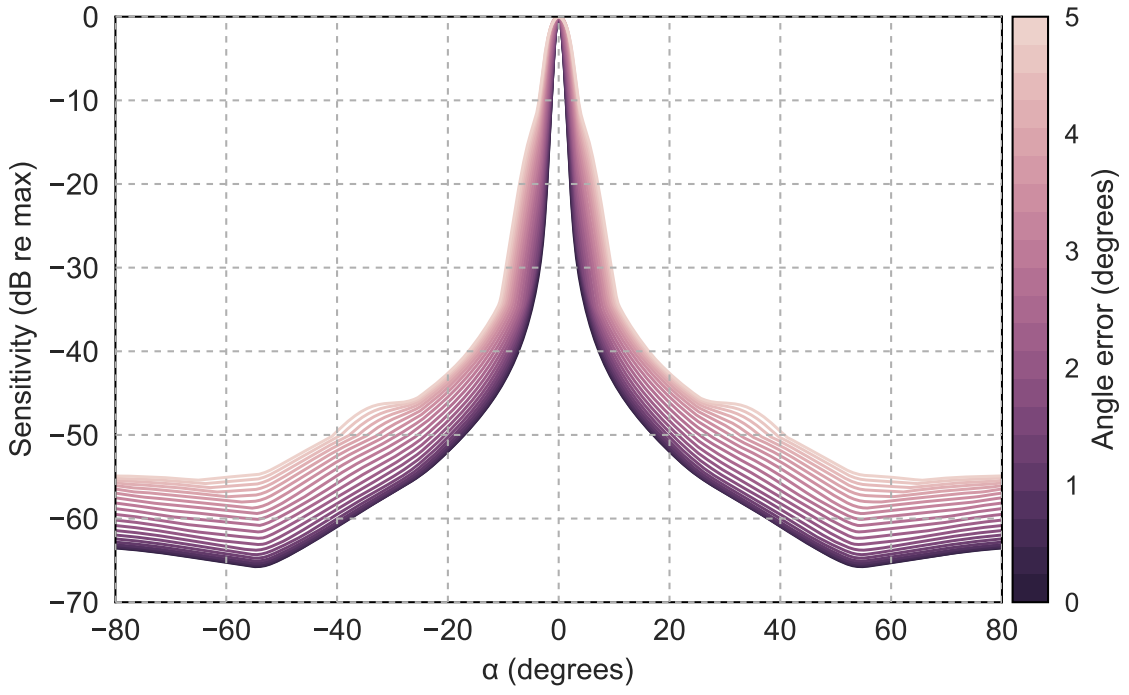
5.6.1 Folding errors

The beamplot in α is shown for acute folding errors in Fig. 70. When plotted on an absolute scale, the error appears to increase near and far side-lobe levels by a small amount (around 2–3 dB). However, peak sensitivity also decreases by around 10 dB. The relative side-lobe change is seen clearly when the beamplots are plotted on a relative scale. The change is around 15 dB, accompanied by a change in structure with emphasis on far side-lobe levels. Main-lobe width appears to increase by 1–2° due to the decrease in aperture size along the x-dimension. The beamplot in β, shown in Fig. 71, exhibits the same decrease in peak sensitivity with a more uniform increase in side-lobe levels of about 10 dB.

When obtuse folding errors occur, the result mimicks the acute case. After all, whether the pane is folded towards or away from the focus, the errors in phase based on path-length differences is approximately the same. The beamplots for obtuse folding errors is shown for α (Fig. 72) and β (Fig. 73).

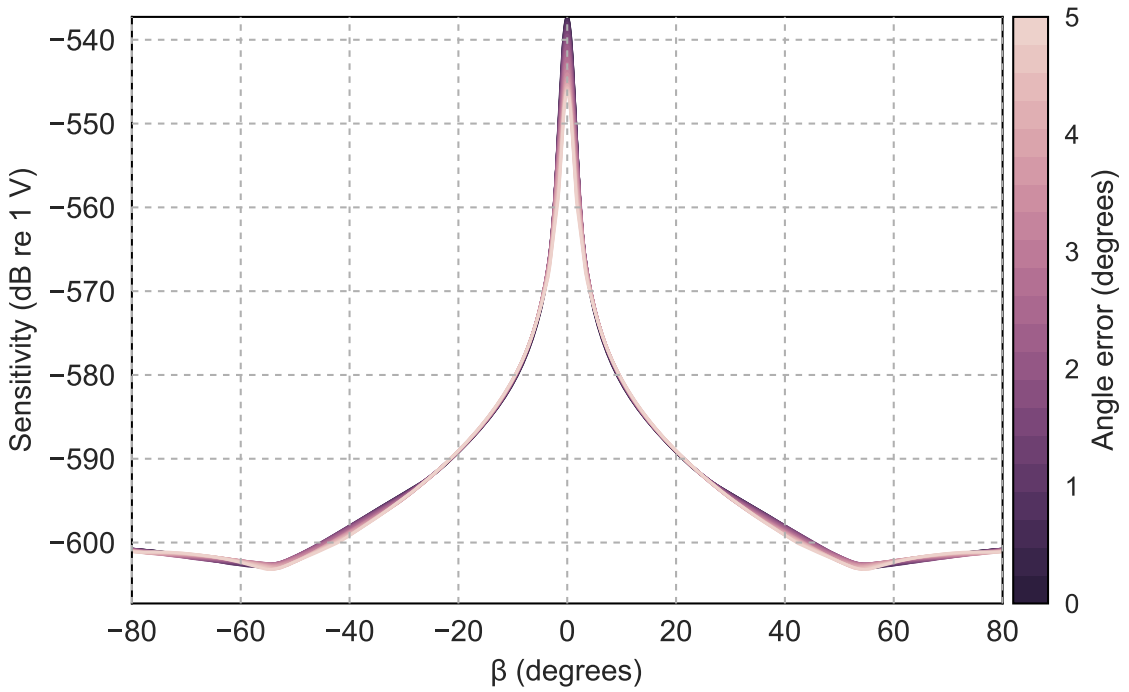


(a) Absolute scale

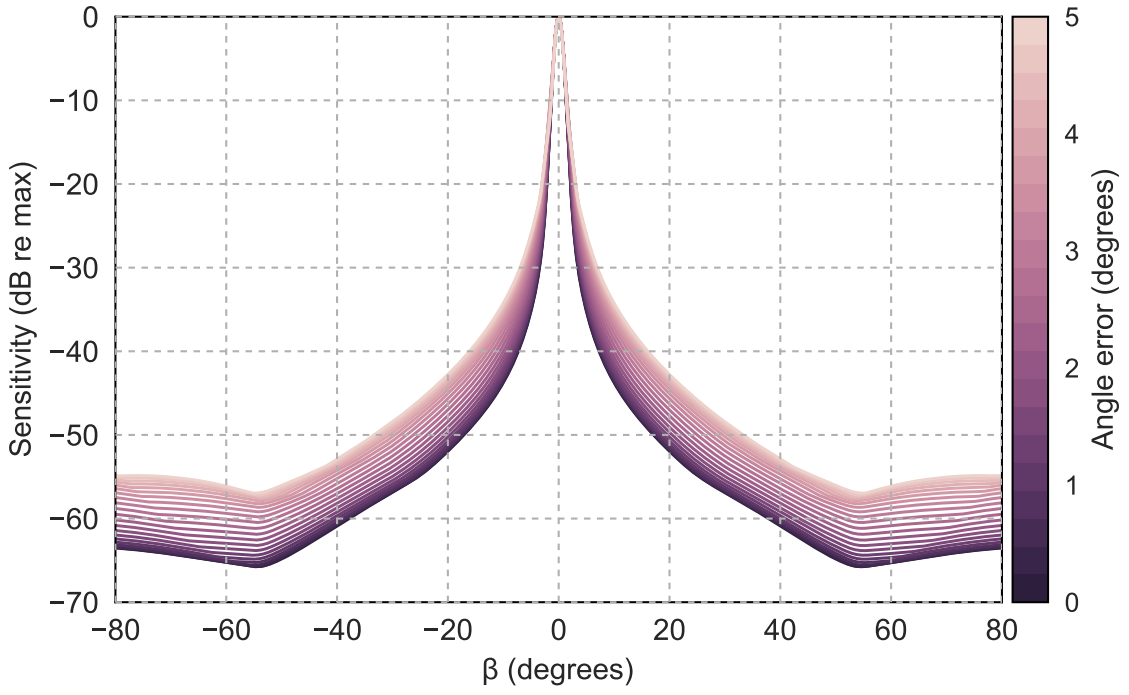


(b) Relative scale

Figure 70: Beamplot degradation along the z-x plane for acute folding errors up to 5°.

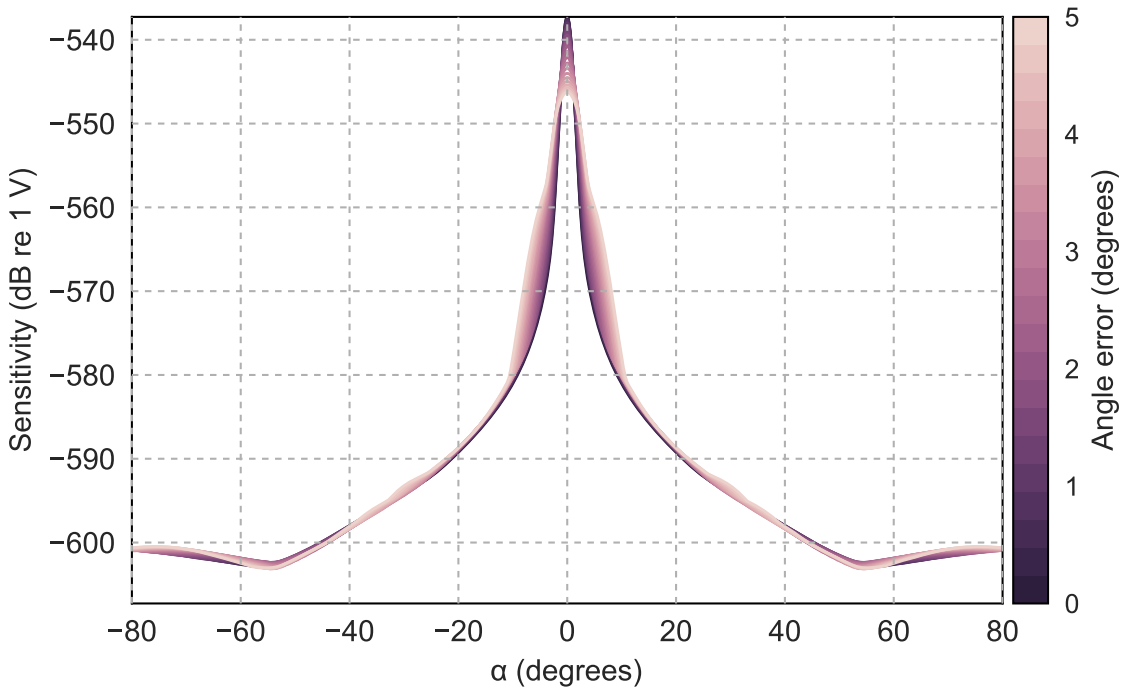


(a) Absolute scale

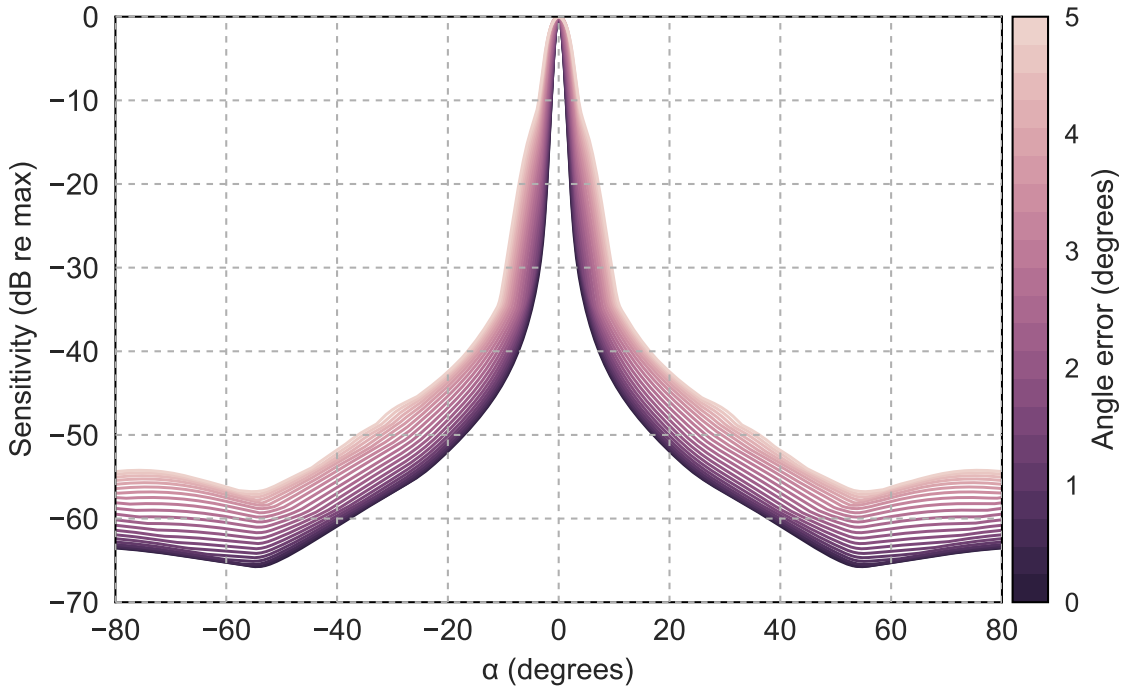


(b) Relative scale

Figure 71: Beamplot degradation along the z-y plane for acute folding errors up to 5°.

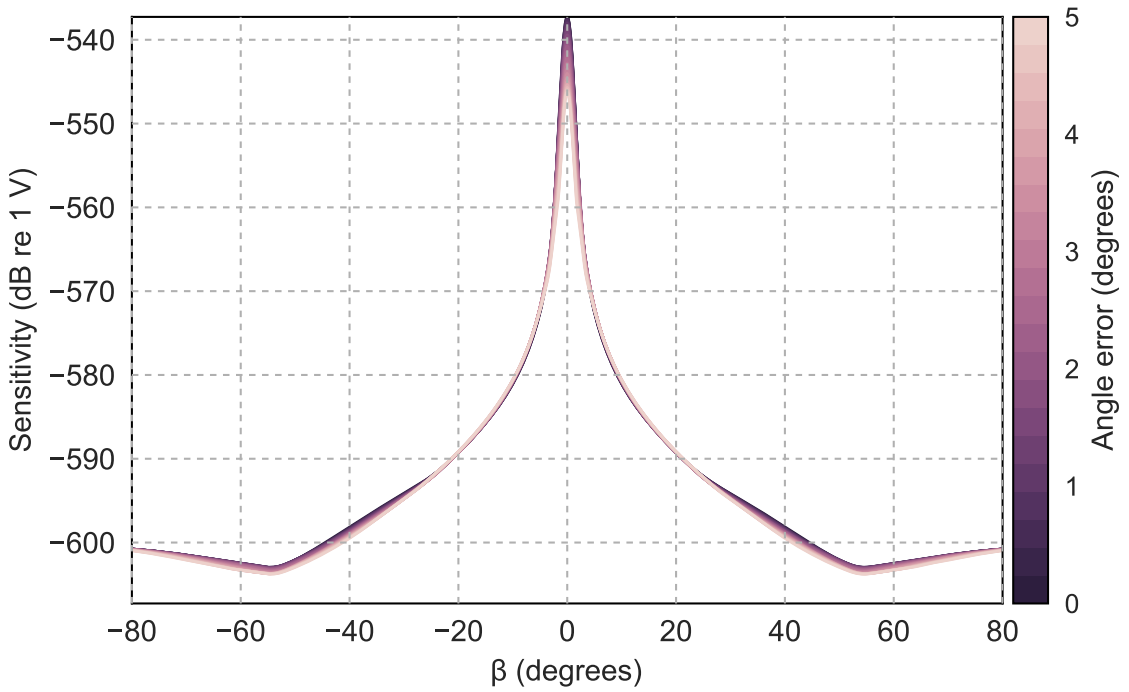


(a) Absolute scale

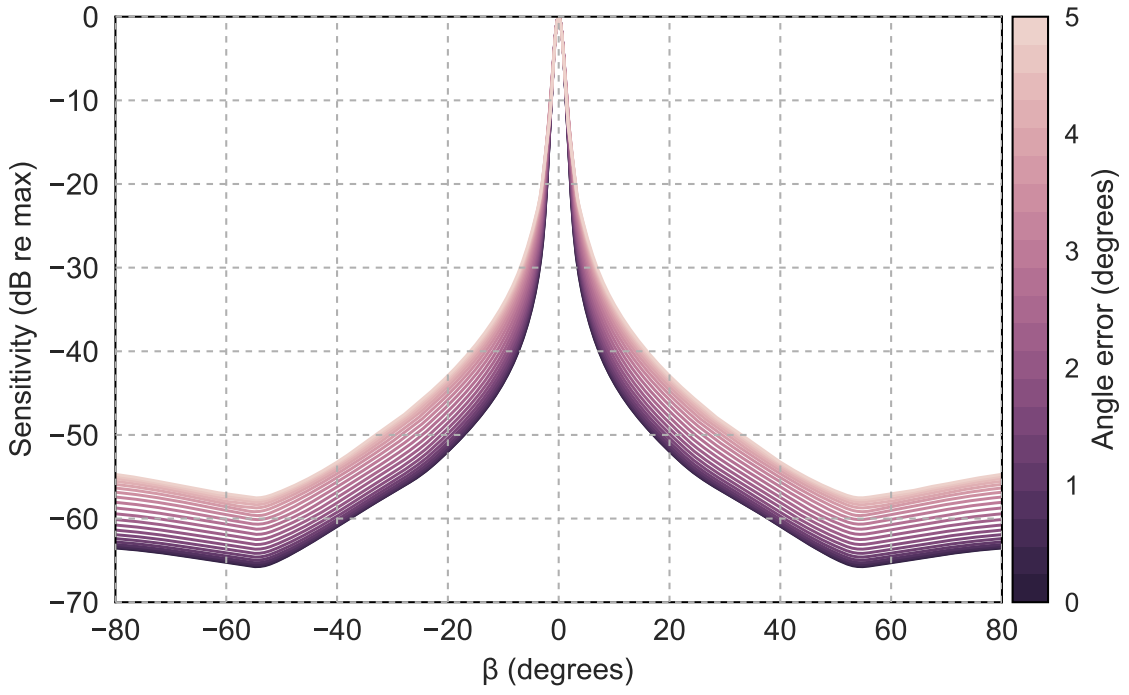


(b) Relative scale

Figure 72: Beamplot degradation along the z-x plane for obtuse folding errors up to 5°.

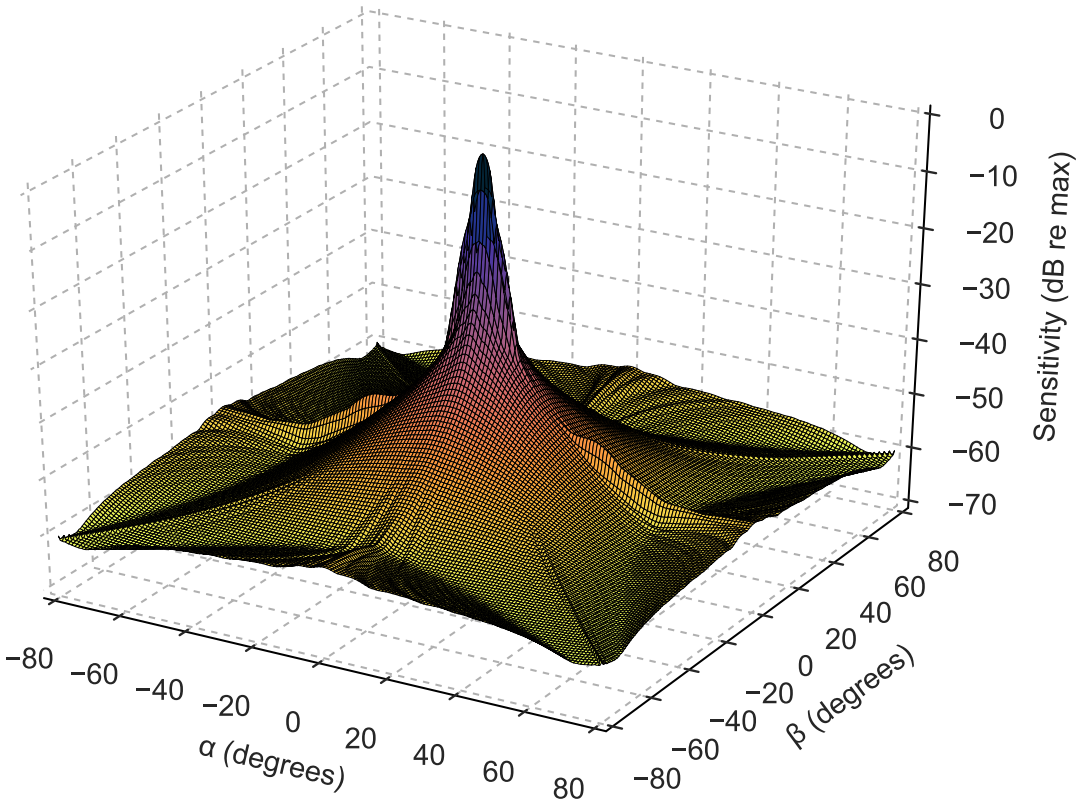


(a) Absolute scale

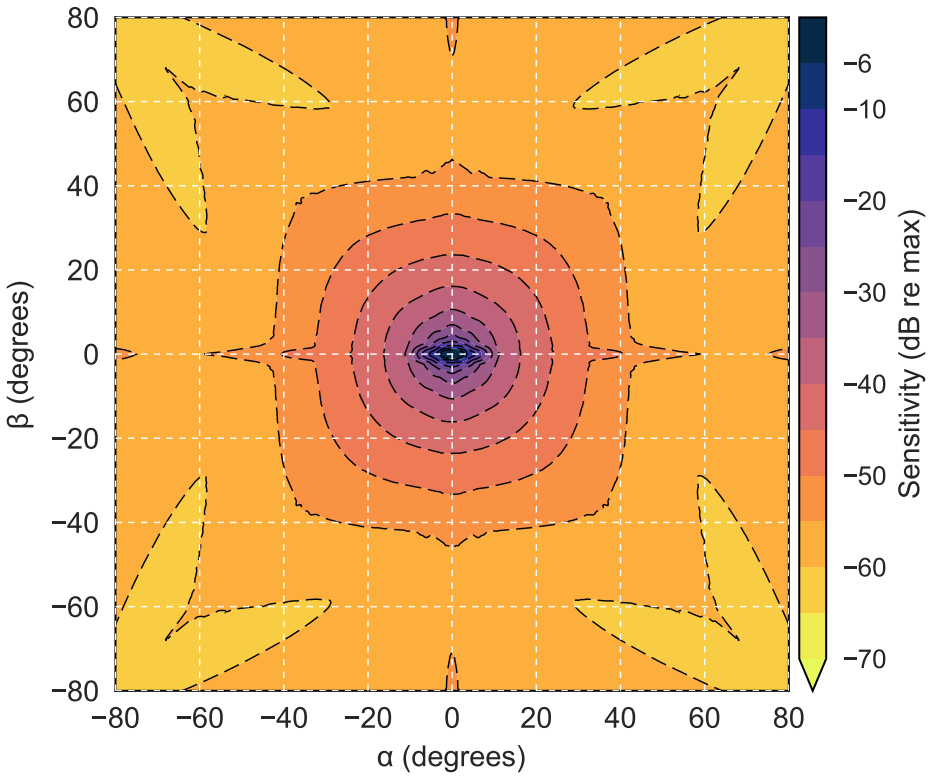


(b) Relative scale

Figure 73: Beamplot degradation along the z-y plane for obtuse folding errors up to 5°.

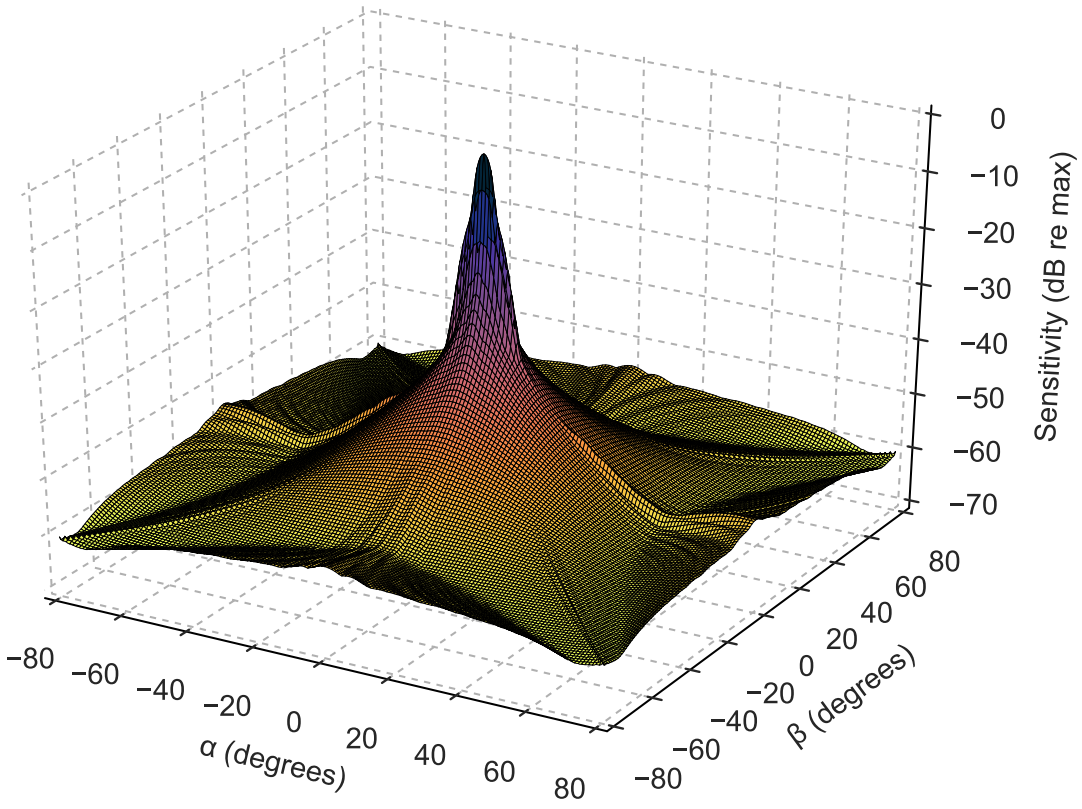


(a) On-axis beamplot surface

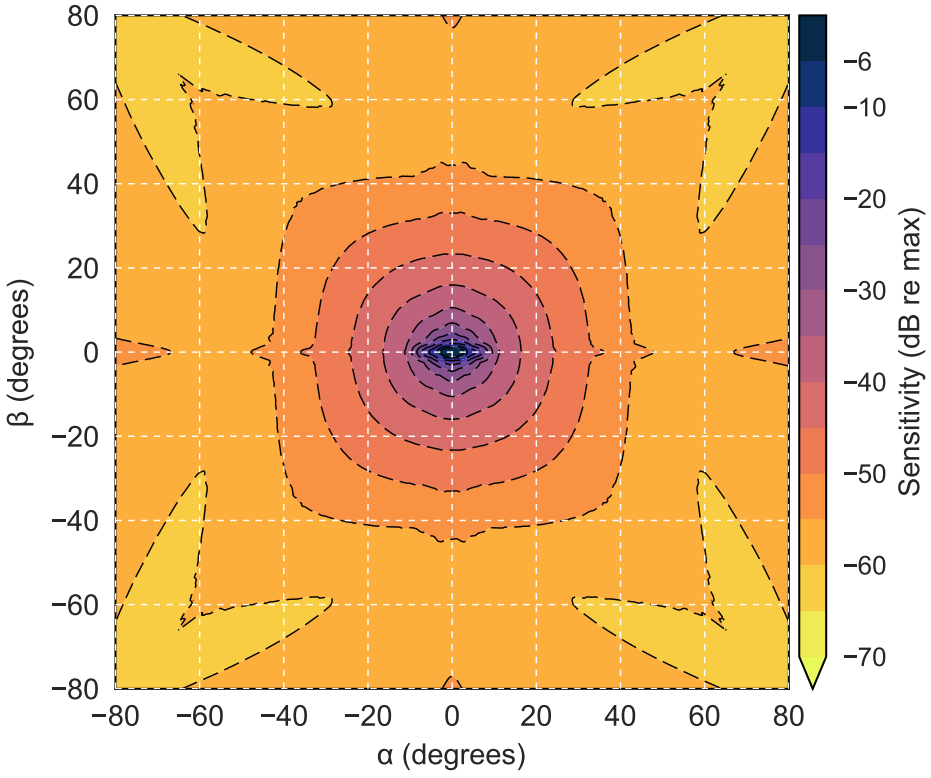


(b) On-axis beamplot contours

Figure 74: Beamplot for 5° acute folding error.



(a) On-axis beamplot surface



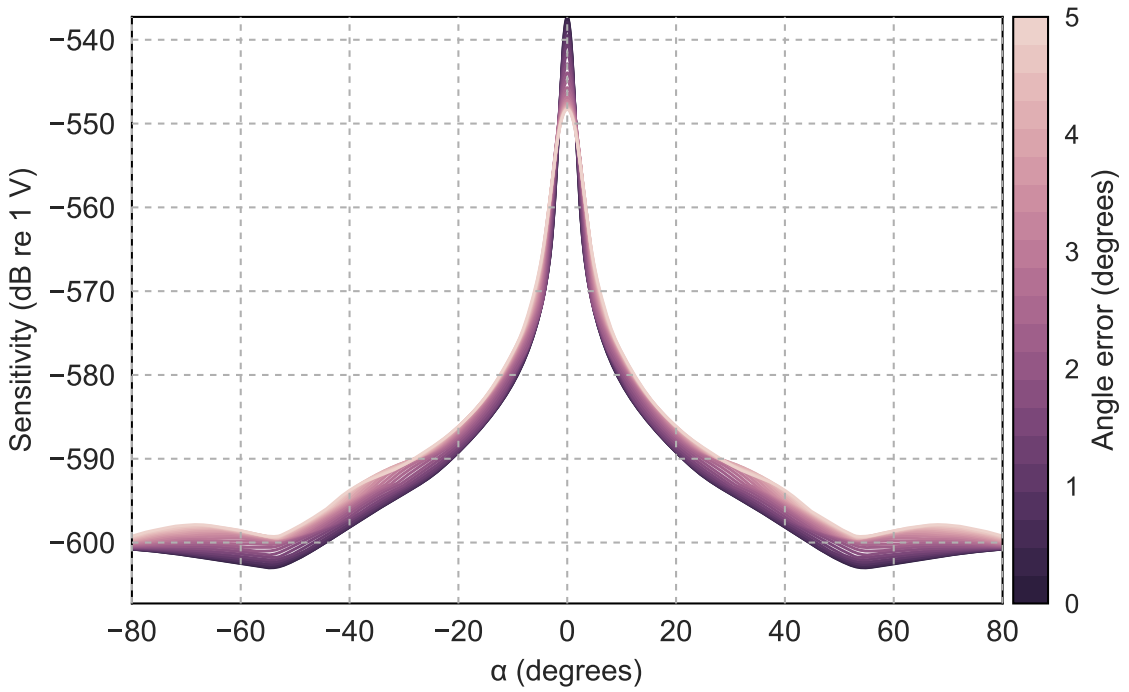
(b) On-axis beamplot contours

Figure 75: Beamplot for 5° obtuse folding error.

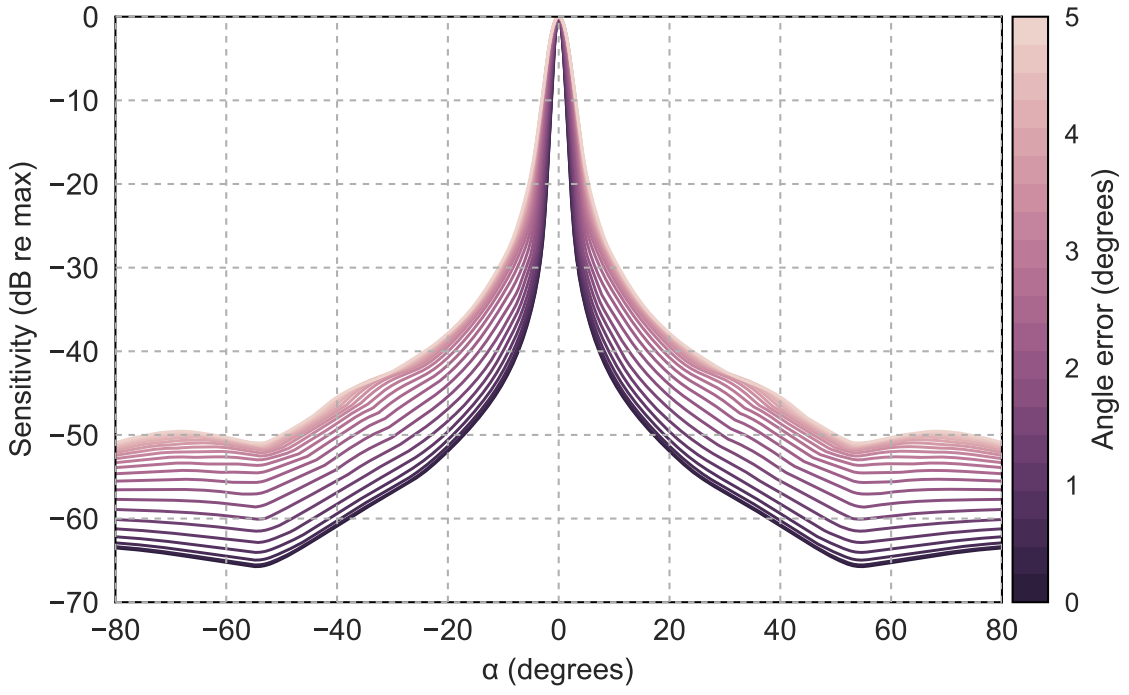
5.6.2 Twist errors

The α and β beamplots are shown in Fig. 76 and in Fig. 77, respectively, for an array experiencing aligned twist errors. In α, we again see a decrease in peak sensitivity of around 10 dB and an increase in side-lobe levels. Here, the side-lobe levels degrade rapidly to a maximum change of 18 dB at 5°. In β, an interesting phenomena occurs with the appearance of a secondary phantom peak. The phantom peak occurs because the two side panes which normally direct energy (approximately) along their normal are now directed off-axis by the error angle. In an image, the secondary peak will create an undesirable phantom image artifact. Side-lobe levels are observed to increase by around 14 dB.

When the side panes twist in opposing directions, the change in the beamplot along α mimicks the aligned case (see Fig. 78. In β, however, each side-pane now directs energy in a different direction, creating two phantom peaks with lower energy. This is shown in Fig. 79.

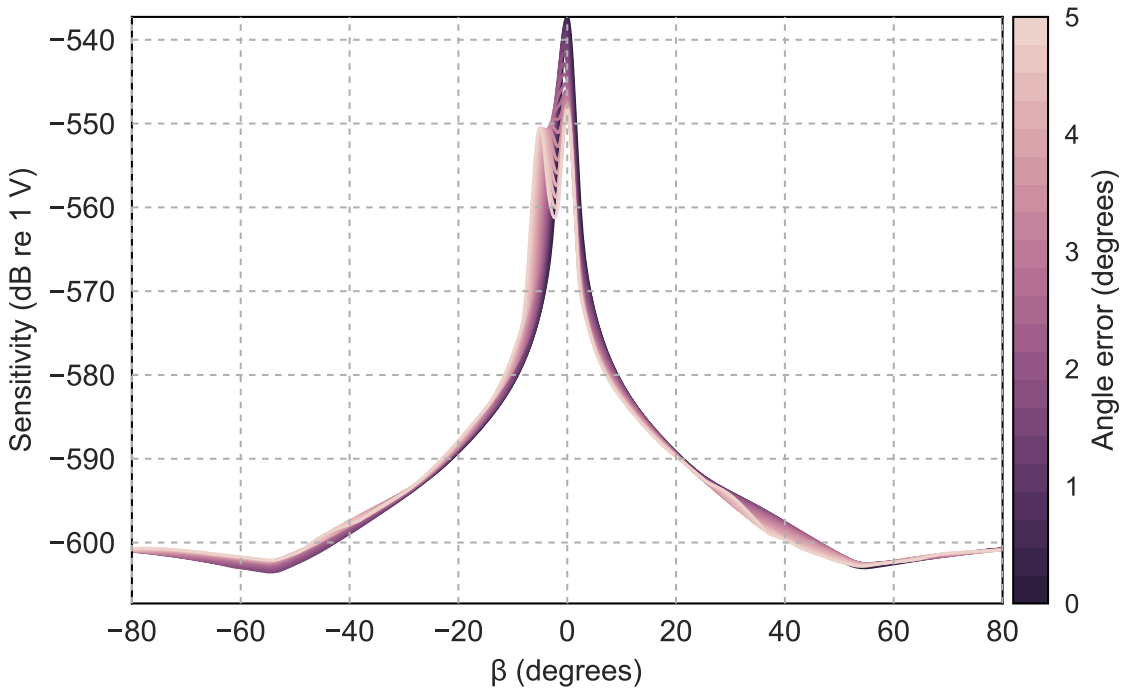


(a) Absolute scale

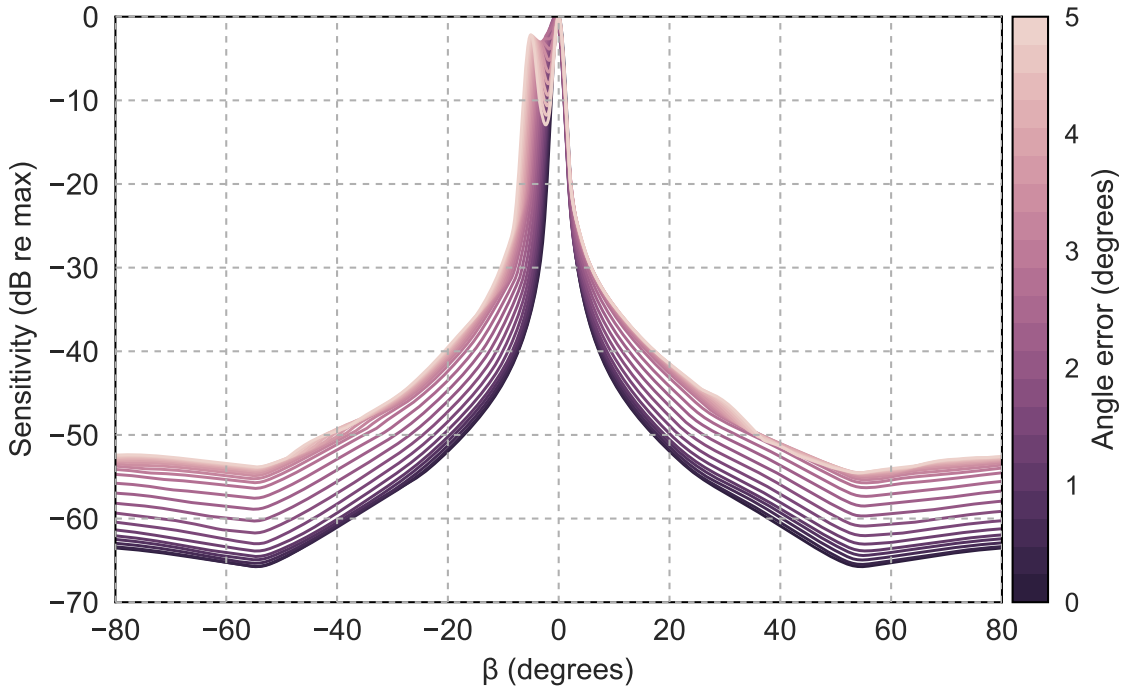


(b) Relative scale

Figure 76: Beamplot degradation along the z-x plane for aligned twist errors up to 5°.

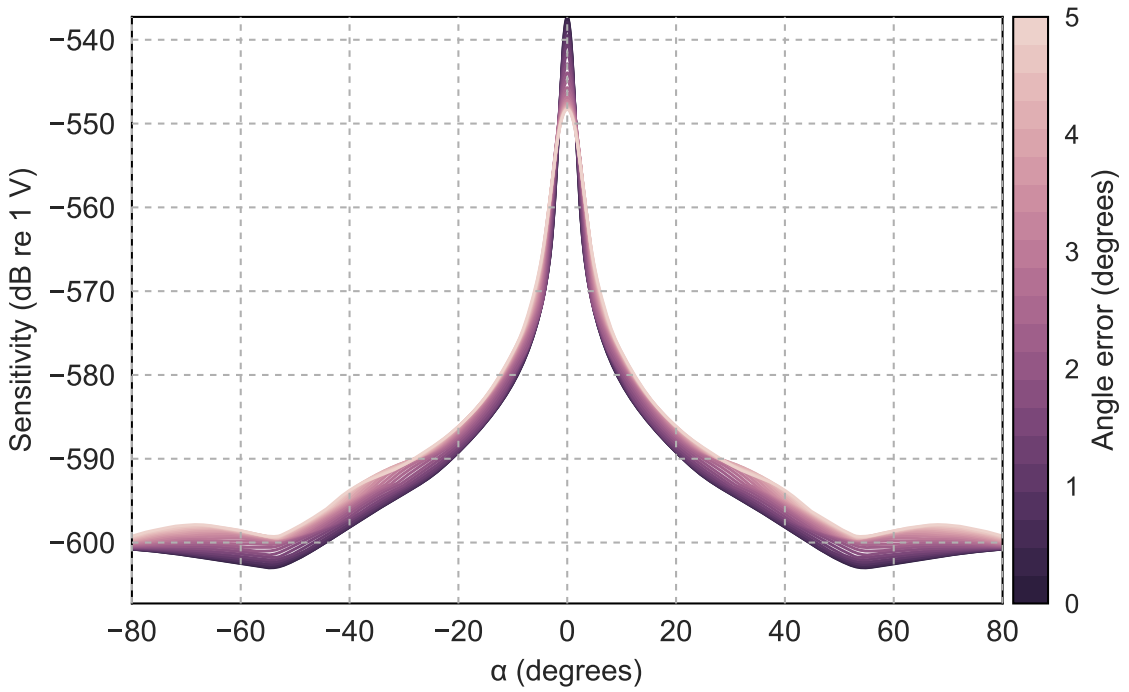


(a) Absolute scale

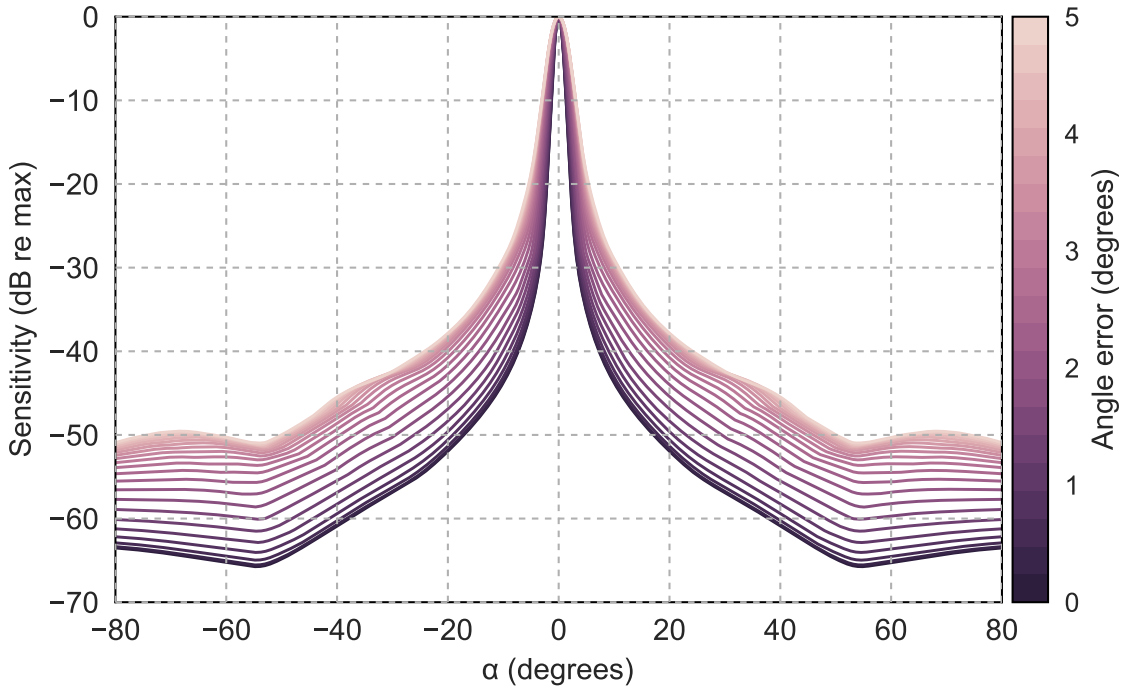


(b) Relative scale

Figure 77: Beamplot degradation along the z-y plane for aligned twist errors up to 5°.

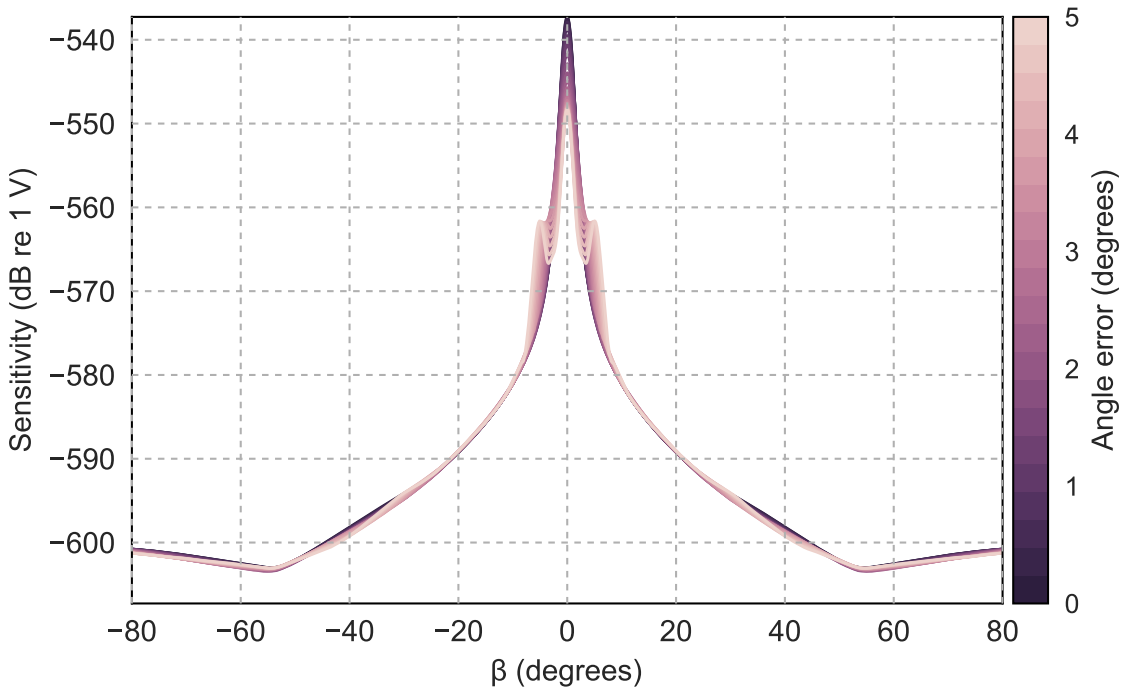


(a) Absolute scale

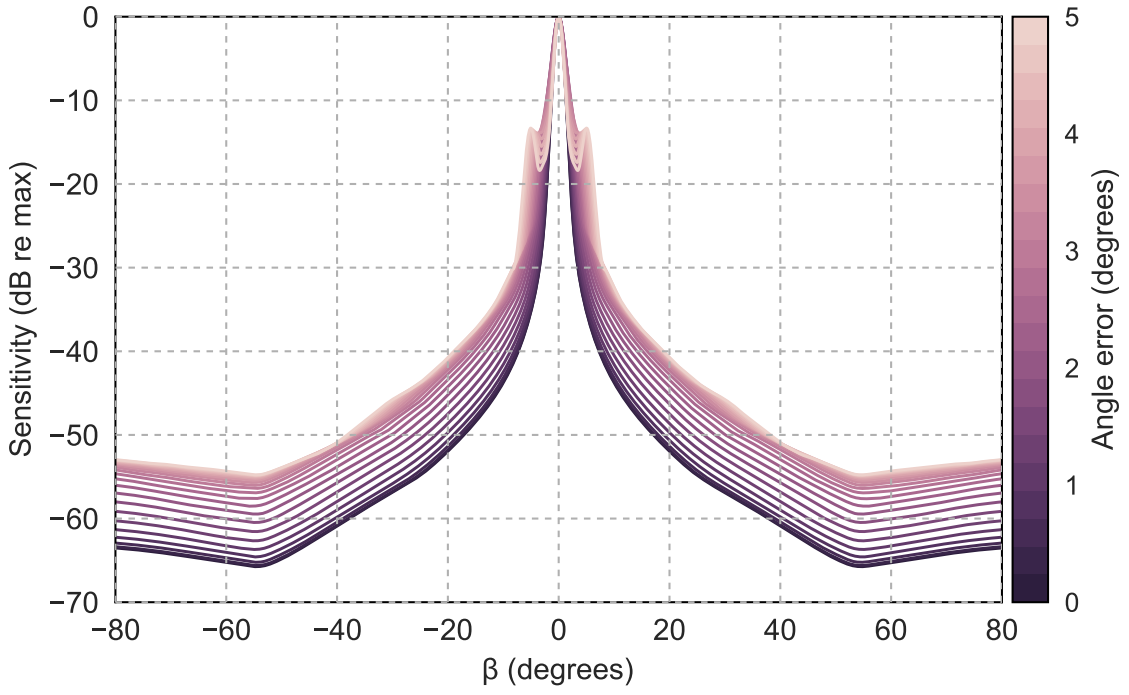


(b) Relative scale

Figure 78: Beamplot degradation along the z-x plane for opposed twist errors up to 5°.

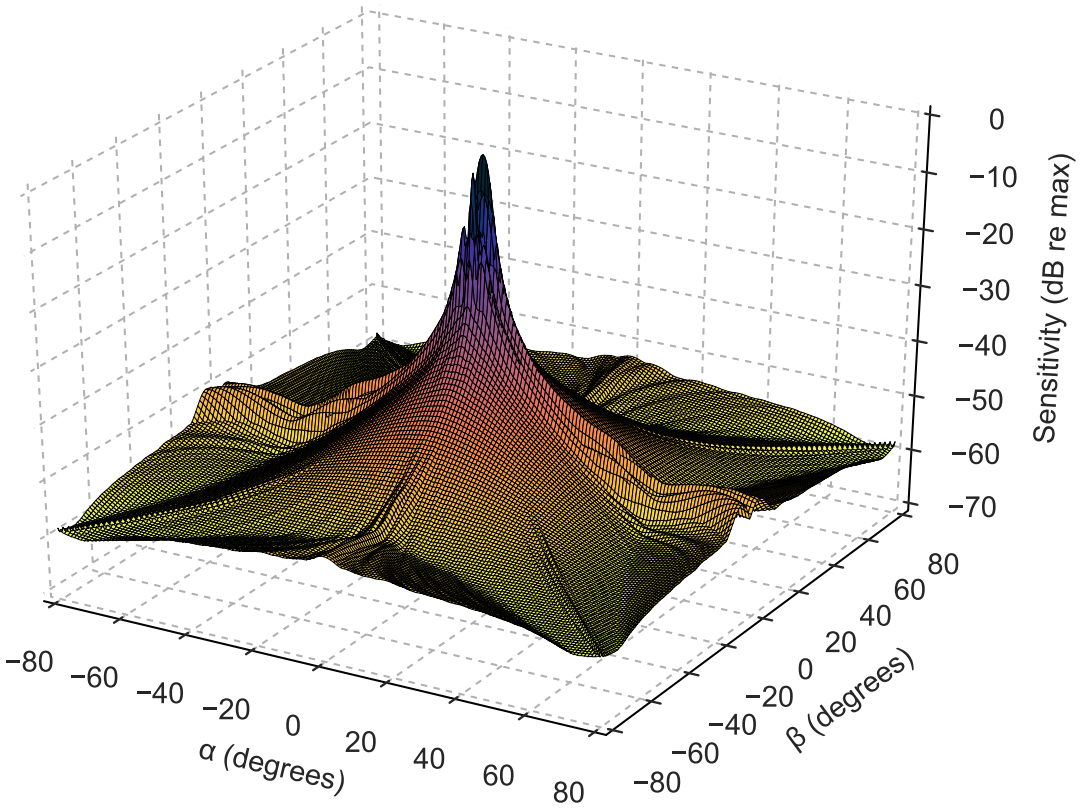


(a) Absolute scale

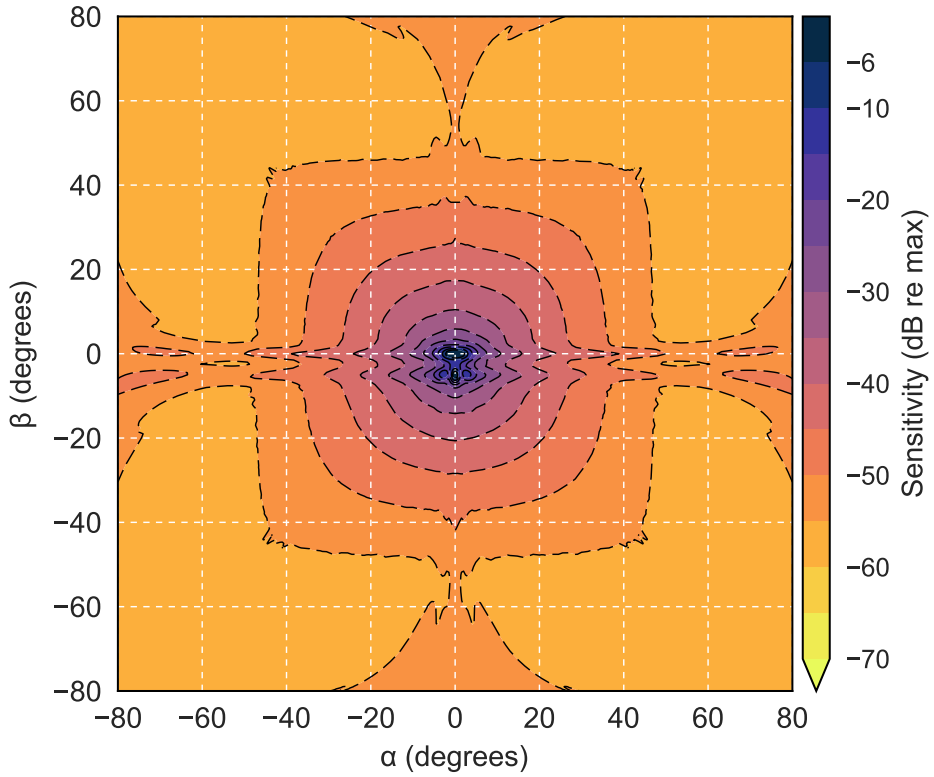


(b) Relative scale

Figure 79: Beamplot degradation along the z-y plane for opposed twist errors up to 5°.

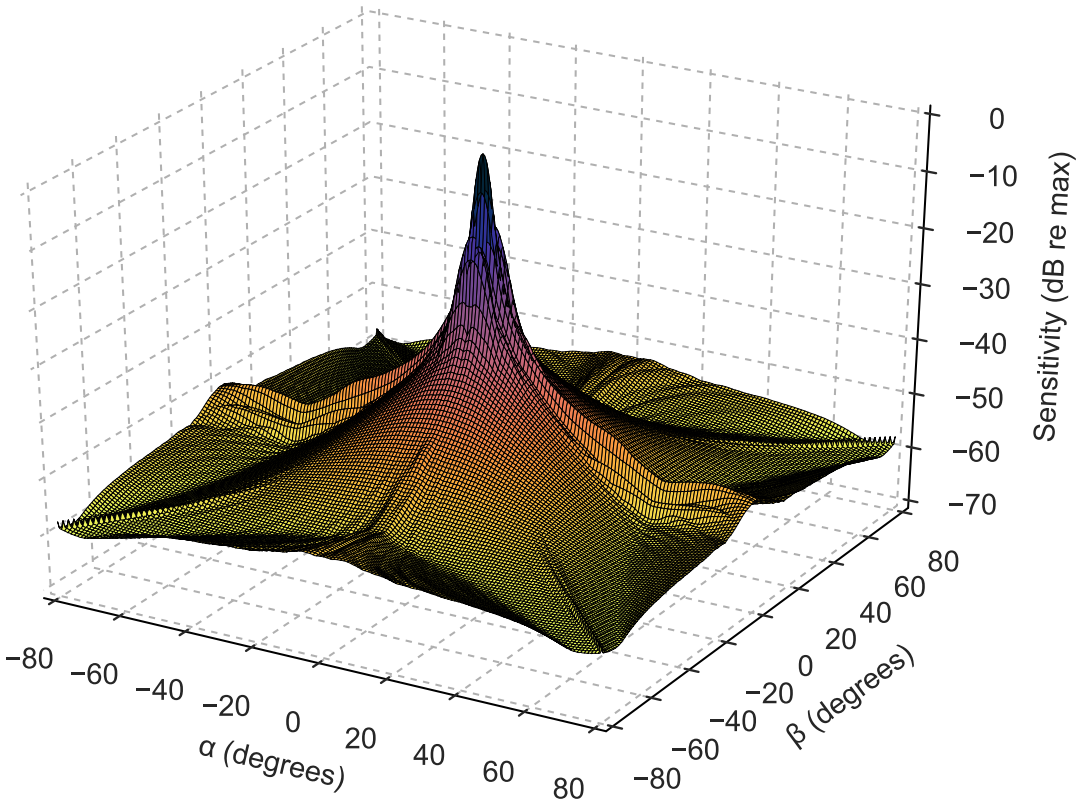


(a) On-axis beamplot surface

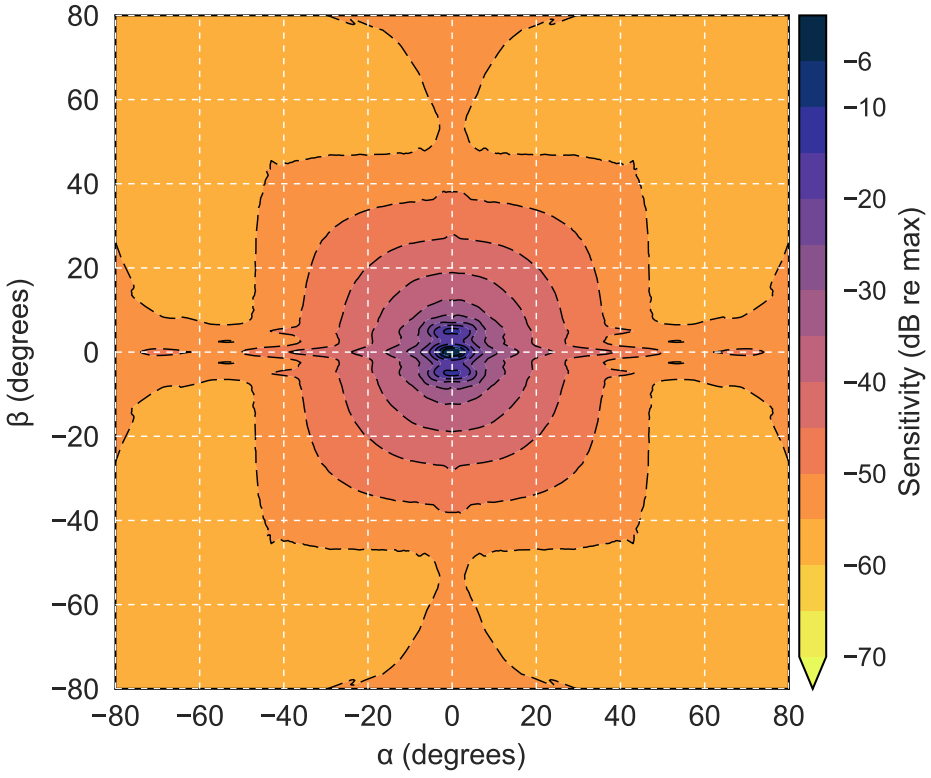


(b) On-axis beamplot contours

Figure 80: Beamplot for 5° aligned twist error.



(a) On-axis beamplot surface



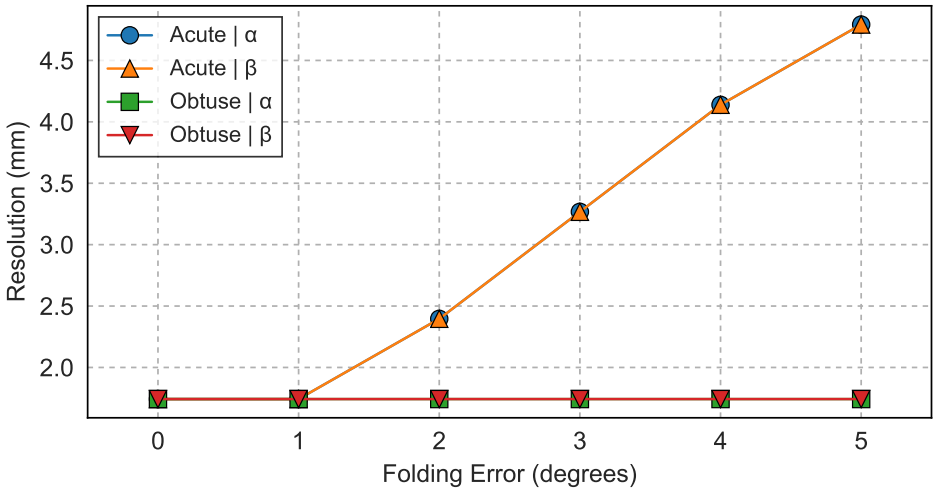
(b) On-axis beamplot contours

Figure 81: Beamplot for 5° opposed twist error.

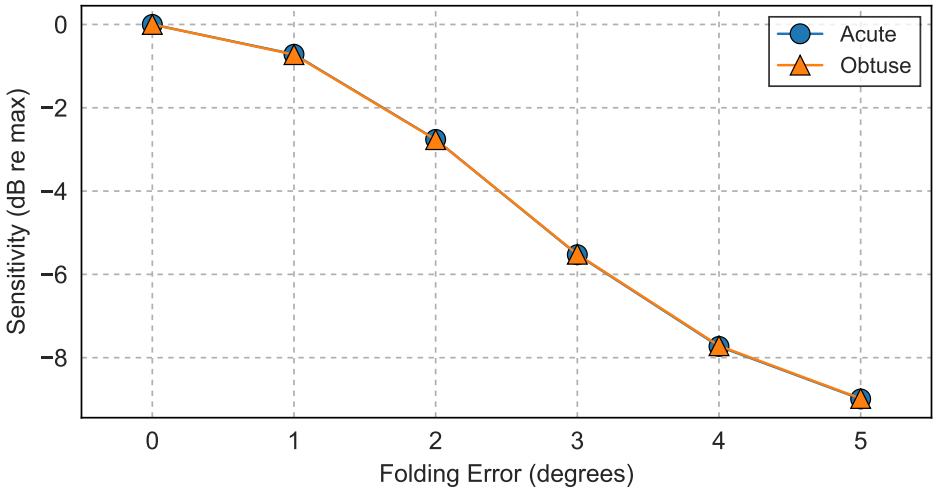
5.6.3 Performance trends

To quantify the observed changes, resolution, sensitivity and CTR were calculated from the degraded beamplots as a function of the error angle. For acute and obtuse folding error, these trends are shown in Fig. 82. Resolution in α is found to remain about the same up to 1° and then to rapidly increase for larger errors, most likely due to the fact that the aperture size in x decreases with folding angle. In β, because the aperture size in y is unaffected, the resolution remains the same. CTR is observed to increase nearly in line with the drop in sensitivity. The results indicate that folding error limited mechanically to within $\pm 1^\circ$ will yield no change in the resolution, and less than 1 dB degradation of both CTR and sensitivity. For error above 2°, the imaging performance will be severely compromised.

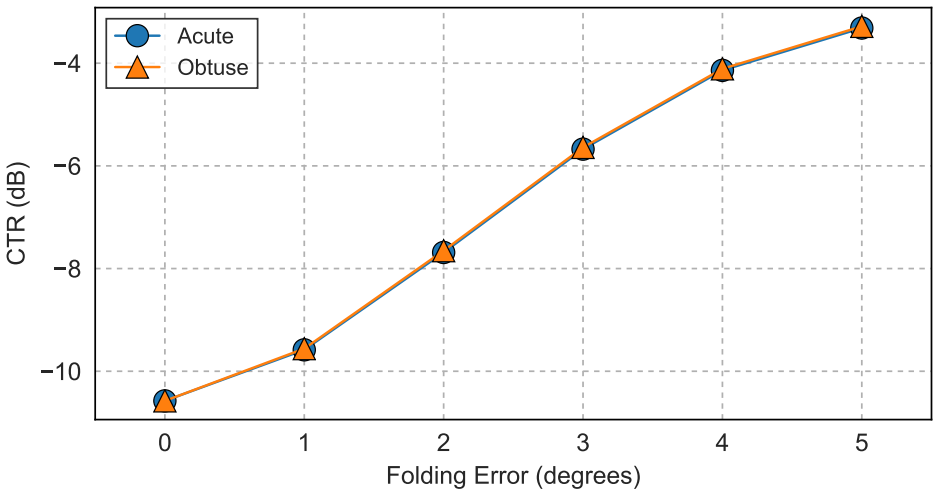
For aligned and opposed twist error, performance trends are shown in Fig. 83. When the twist error is aligned, the resolution trends in both α and β indicate a small increase at 1° followed by a rapid increase above 1°. When the twist error is opposed, the resolution trends reach a peculiar maximum at 3° for α and 2° for β. In fact, these angles mark the onset of phantom peaks in the beamplots which distorts the resolution calculation. Both CTR and sensitivity degrade rapidly, with the higher rate of decay observed for the case of opposed twist errors.



(a) Detail resolution

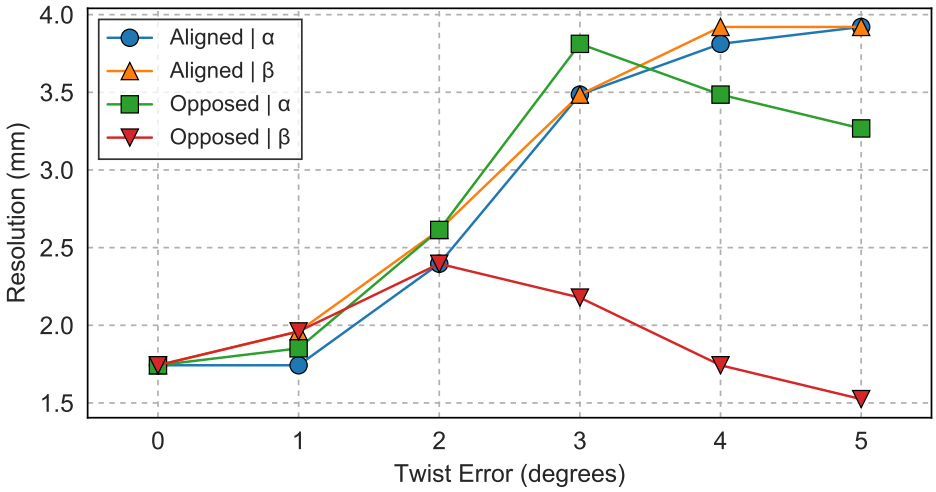


(b) Sensitivity

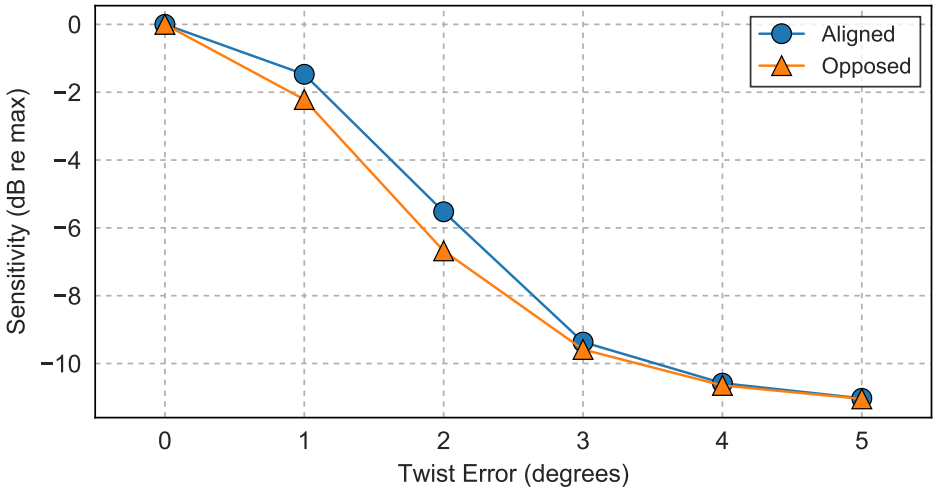


(c) Contrast resolution

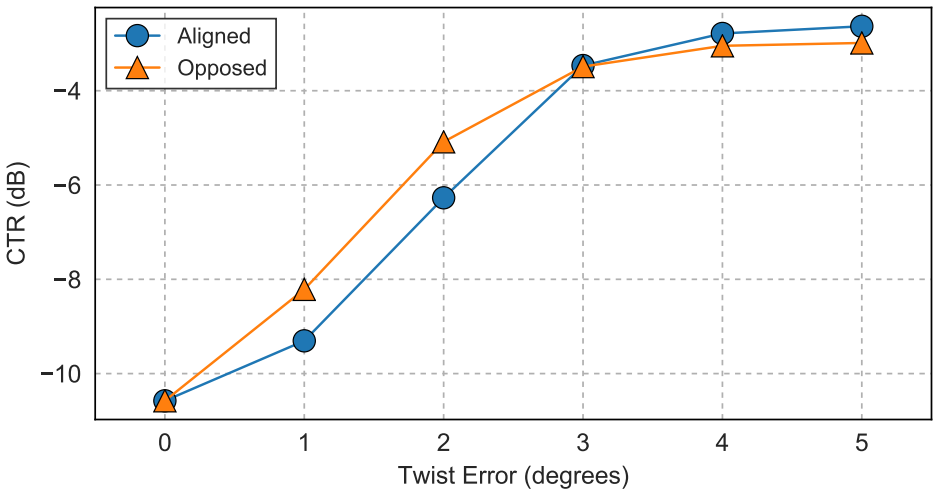
Figure 82: Performance measure trends as a function of increasing folding error.



(a) Detail resolution



(b) Sensitivity



(c) Contrast resolution

Figure 83: Performance measure trends as a function of increasing twist error.

5.6.4 Error correction scheme

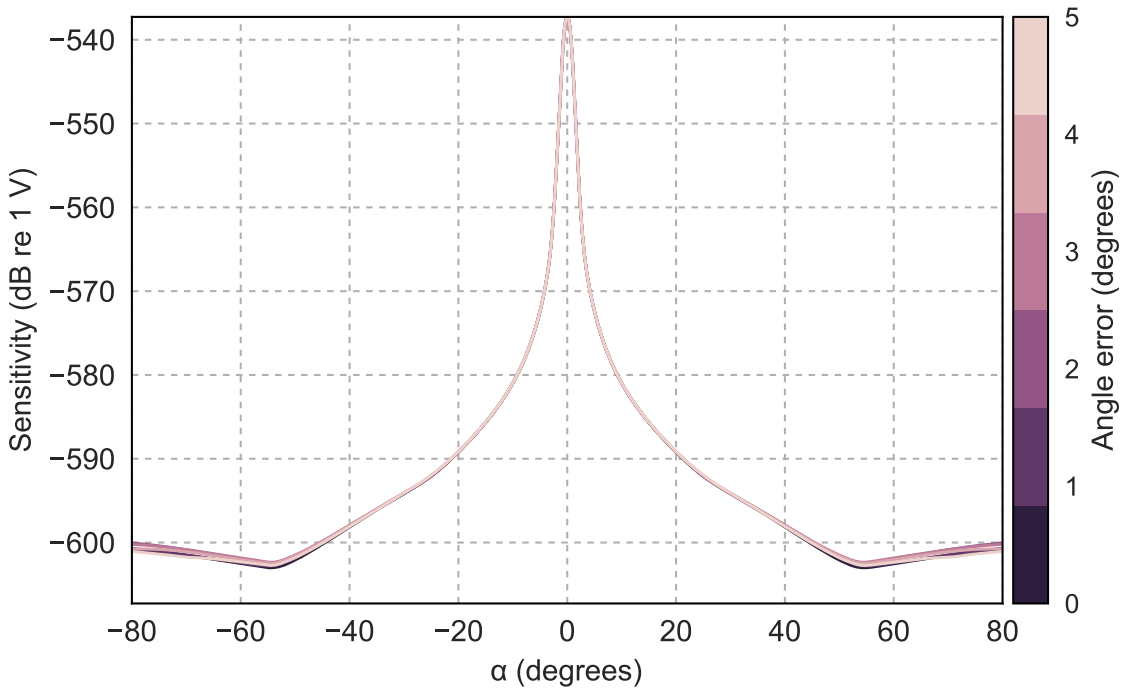
Modern catheter navigation systems are capable of precise real-time tracking of catheter tip location and orientation. For example, CARTO (Biosense Webster) utilizes static magnetic fields of varying strength and embedded inductive coil sensors to determine catheter tip location as well as roll, pitch, and yaw orientation. The precision of this particular system is within $\pm 0.1^{\circ}$. It is conceivable that inductive coils may be embedded within each pane of a foldable array (by means of surface micromachining for example) for use in such a navigation system. Precise information regarding relative orientations of the pane could be used in real-time to correct for orientation errors.

To determine the potential benefit of such a system, beamplots were generated for varying folding and twist errors assuming the orientations of the panes were known within $\pm 0.1^{\circ}$. Specifically, for a known error of angle θ, beamplots were generated for $\theta + 0.1^{\circ}$ and $\theta - 0.1^{\circ}$ and the worst of either case was recorded.

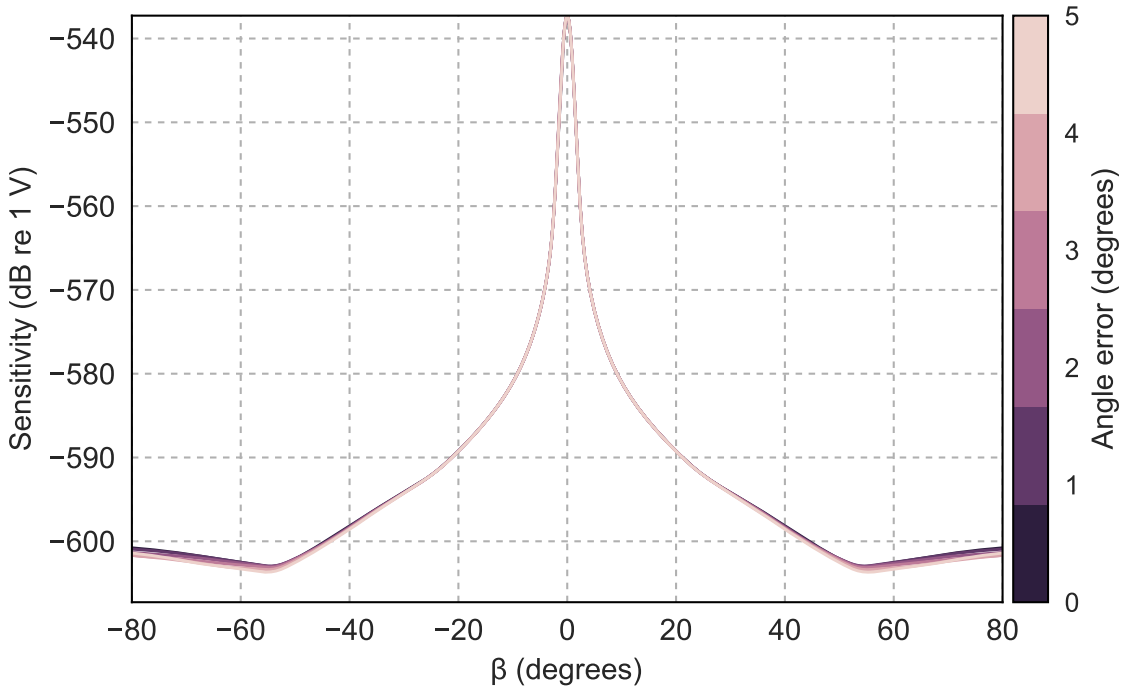
Slices of the generated beamplots along the z-x plane and z-y plane are shown for acute folding errors (Fig. 84), aligned twist errors (Fig. 85), and opposed twist errors (Fig. 86). Obtuse folding errors are not shown since it was determined previously that acute and obtuse folding errors result in similar degradation. In all cases, resolution appears to be completely recovered by correction of the beamformation delays. Side-lobe degradation is like-wise suppressed in all cases except those in the z-x plane for aligned and opposed twist error. In the latter cases, the degradation is limited to less than 5 dB.

As before, to quantify the observed degradation, resolution, sensitivity and ctr were calculated from the beamplots as a function of increasing error angle. For acute folding errors with correction, these trends are shown in Fig. 87. Resolution is found to remain nearly constant, while sensitivity and CTR actually experience a very small improvement, explained by the fact that the folding of the panes brings them slightly closer to the focus. However, because the observed improvement is inconsequentially small, sensitivity and CTR can be considered practically unchanged.

The performance trends for aligned and opposed twist errors with correction are shown in Fig. 88. The trends indicate that resolution is successfully recovered at all angles and that sensitivity and CTR are degraded by less than 0.02 dB and 0.03 dB respectively. These encouraging results suggest that implementation of navigational and orientation tracking for a foldable array can completely remove phase aberrations due to orientation errors up to 5°.

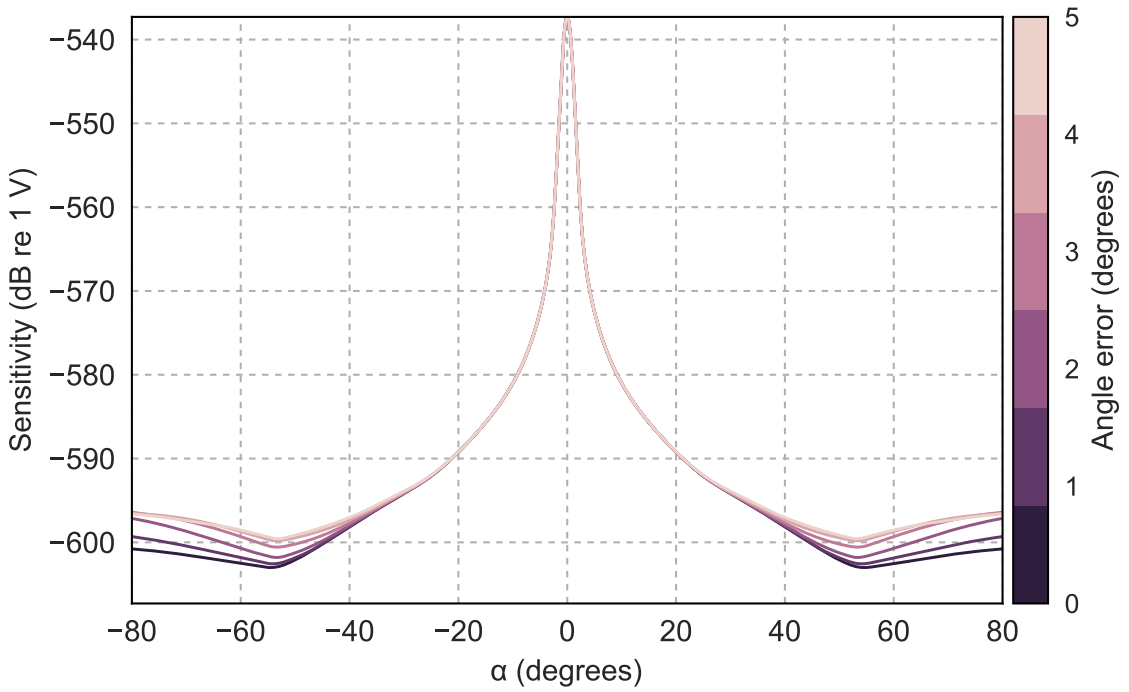


(a) Slice along z-x plane

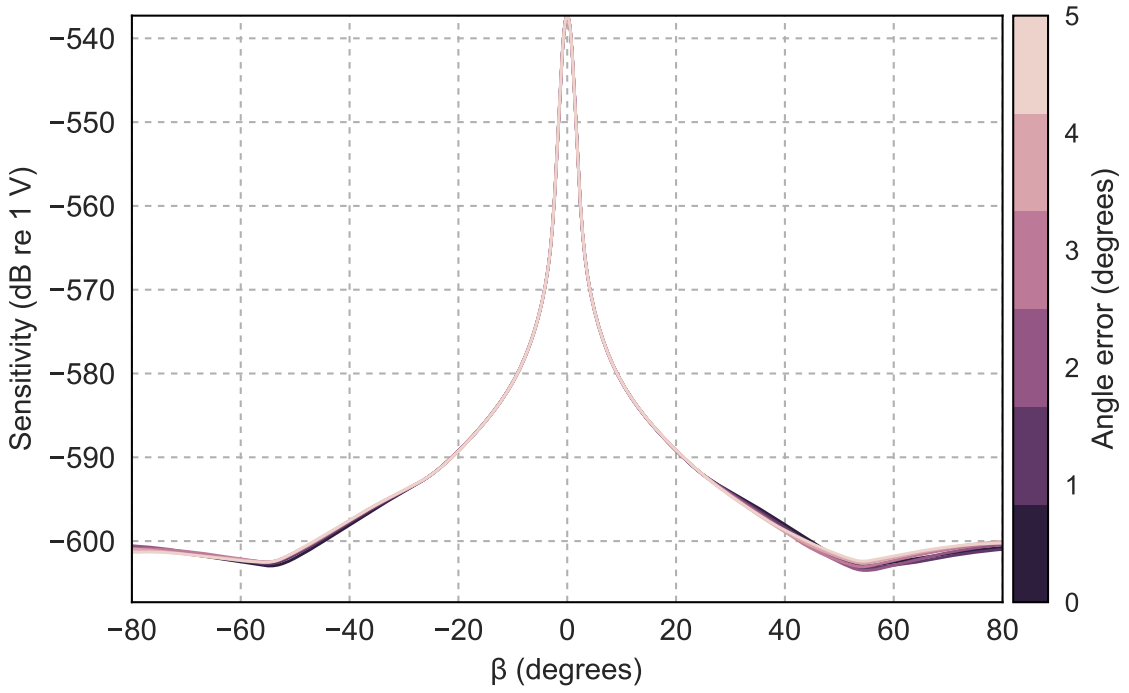


(b) Slice along z-y plane

Figure 84: Beamplot degradation along the z-x and z-y planes after correction within $\pm 0.1^\circ$ for acute folding errors up to 5°.

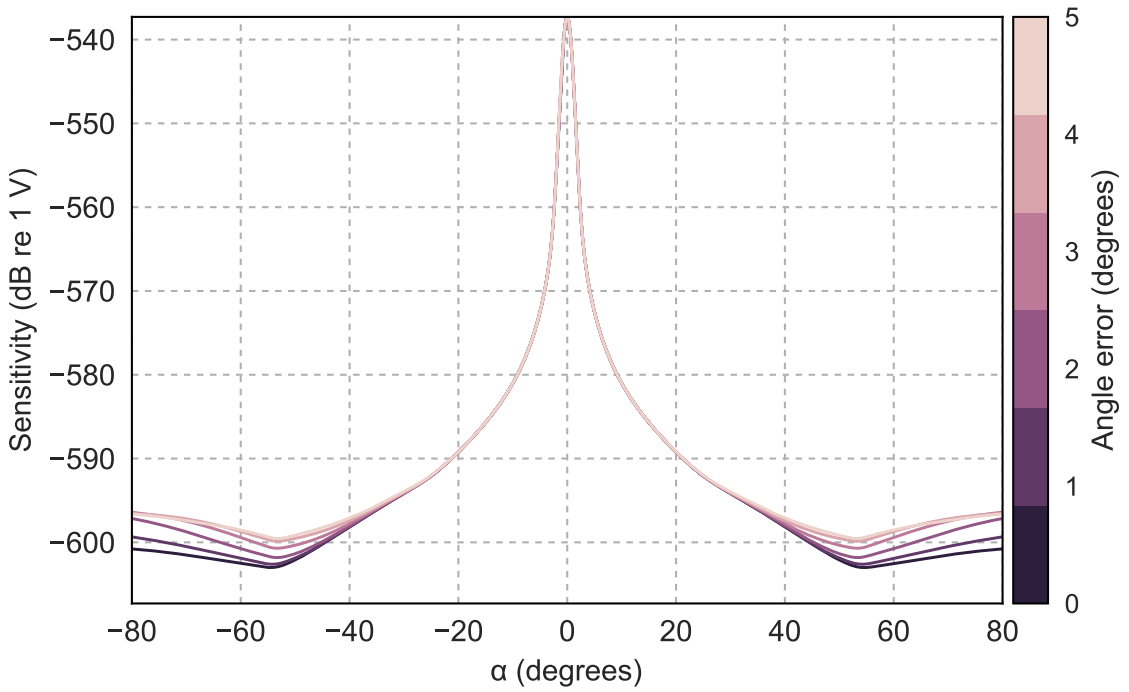


(a) Slice along z-x plane

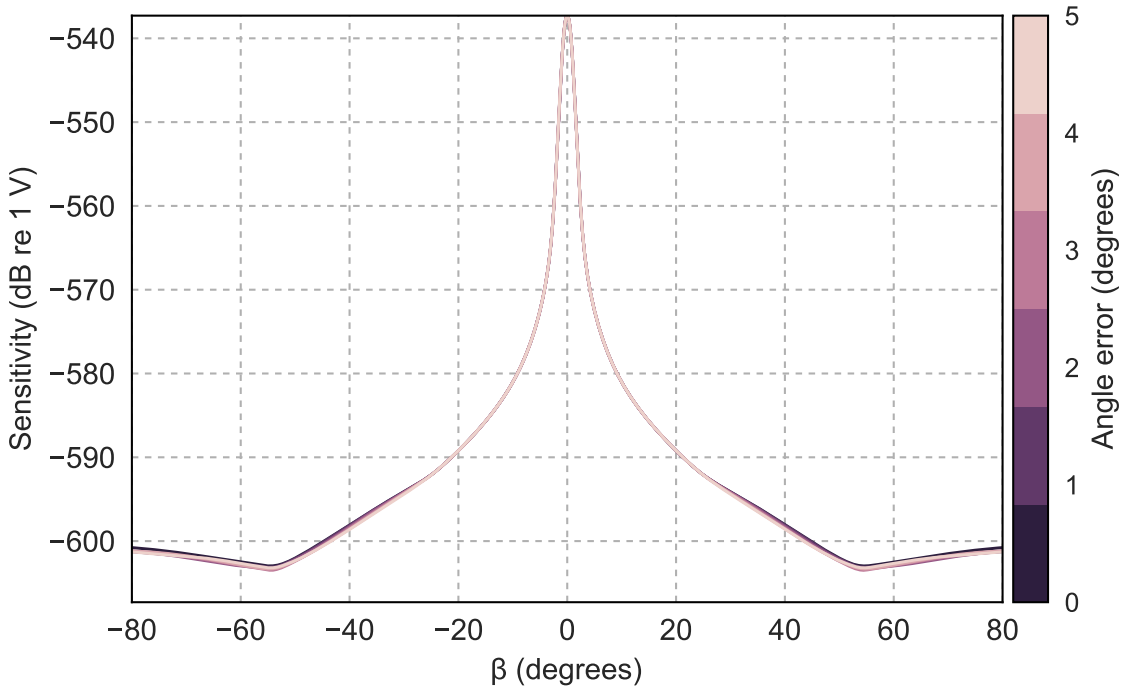


(b) Slice along z-y plane

Figure 85: Beamplot degradation along the z-x and z-y planes after correction within $\pm 0.1^\circ$ for aligned twist errors up to 5°.

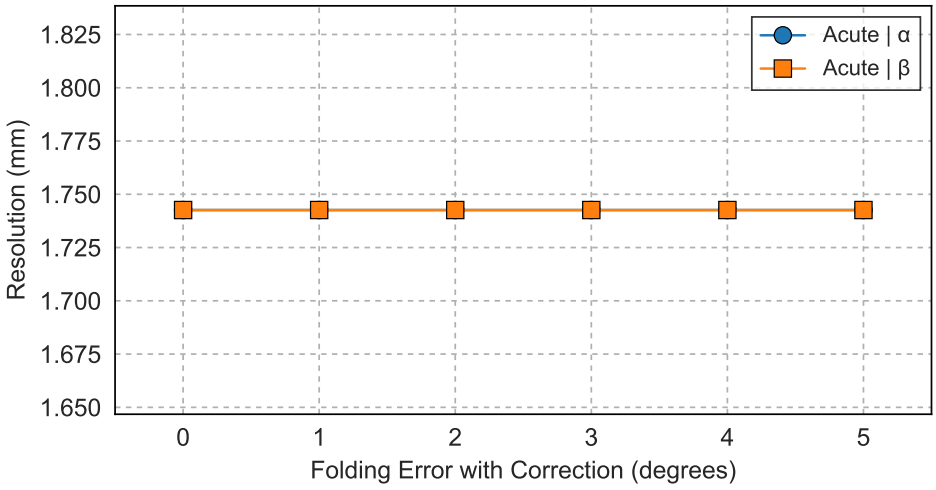


(a) Slice along z-x plane

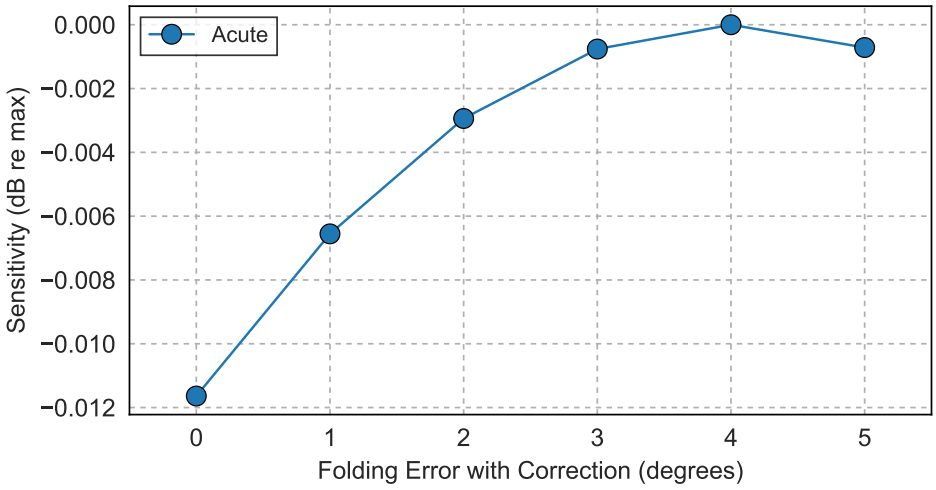


(b) Slice along z-y plane

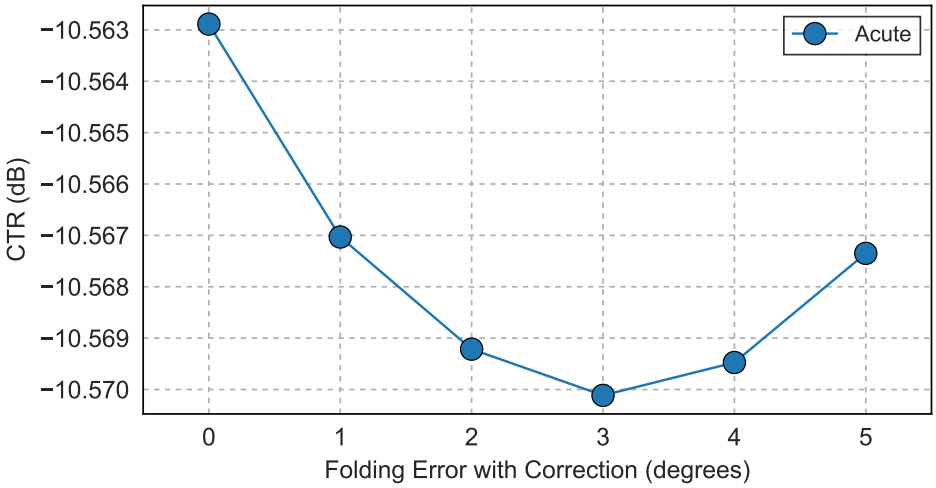
Figure 86: Beamplot degradation along the z-x and z-y planes after correction within $\pm 0.1^\circ$ for opposed twist errors up to 5°.



(a) Detail resolution

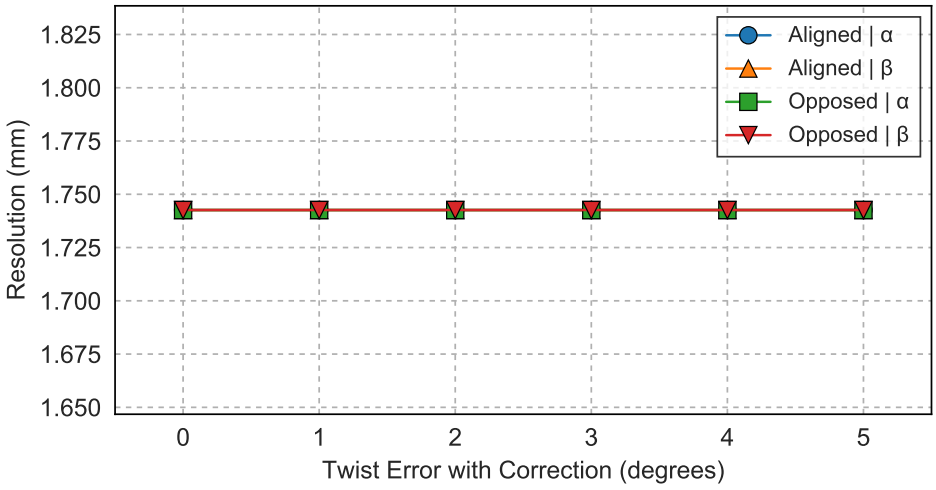


(b) Sensitivity

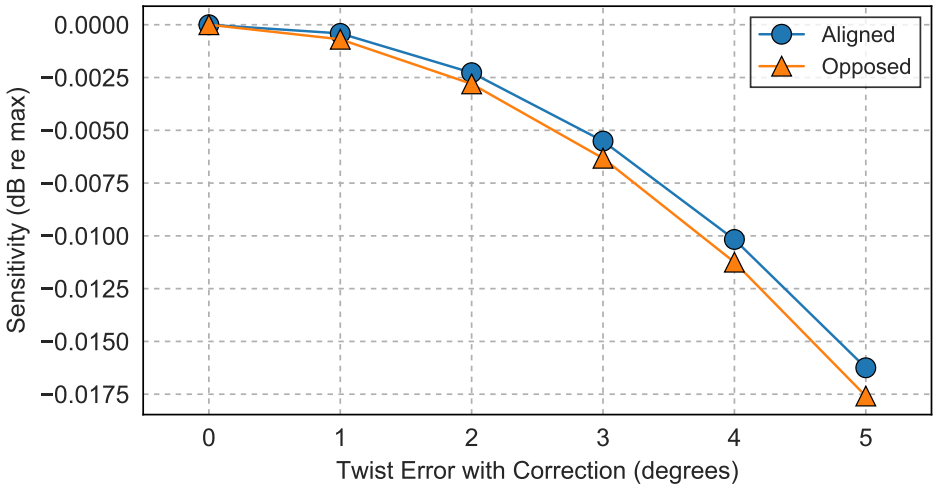


(c) Contrast resolution

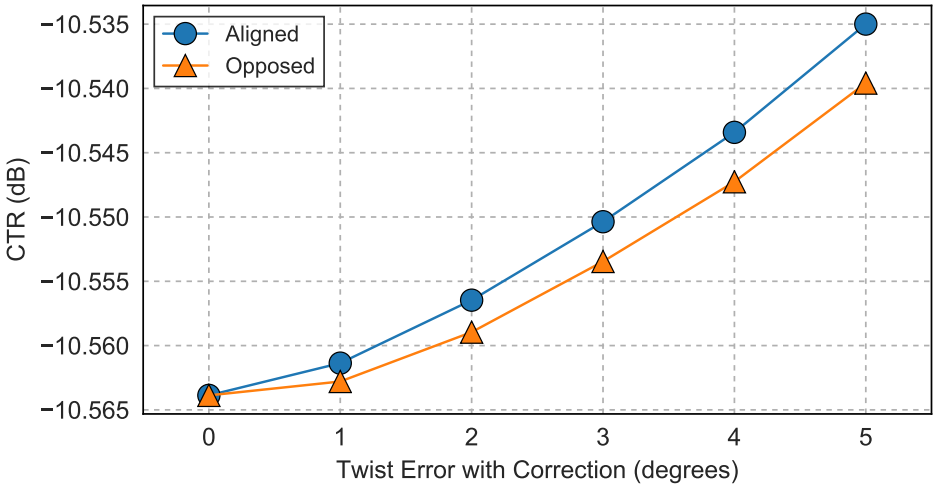
Figure 87: Performance measure trends with correction as a function of increasing folding error.



(a) Detail resolution



(b) Sensitivity



(c) Contrast resolution

Figure 88: Performance measure trends with correction as a function of increasing twist error.

5.7 Summary

A novel foldable 2-D large aperture ICE array based on the CMUT platform was proposed. The array features three foldable panes which can be collapsed to fit within a catheter during delivery into and out of the heart. A sparse array design based on a Vernier concept with 517 transmit and 517 receive elements was chosen to maximize performance with a minimal number of elements. To gauge the feasibility of this design, imaging performance analysis was carried out using a variety of simulation tools. A linear systems model (Field II) was used to calculate two-way beamplots, with CMUT directivity accounted for by an element factor correction. Backscatter from realistic tissue was simulated using a Rayleigh speckle model and a CMUT large-signal transient model to account for non-linear electrostatic force.

Simulated beamplots indicate an on-axis resolution of 1.7 mm and a CTR of -10.23 dB, exceeding the performance targets set. Off-axis performance (40°, 0) was poorer, with a resolution of 2.4 mm and a CTR of -6.23 dB. Backscatter from simulated tissue phantoms indicate that penetration depth in blood and through 1 cm myocardial tissue exceeds the performance target of 15 cm.

A potential risk of a foldable design are phase errors associated with incomplete knowledge of element spatial locations due to incomplete unfolding or mechanical flex. To determine the significance of these orientation errors, beamplots were generated for increasing folding and twist angles. Beamplot degradation indicates that a hinge mechanism constrained to $\pm 1^\circ$ accuracy would be sufficient to maintain adequate performance. Degradation was more susceptible to flex, which should be limited to around $\pm 0.5^\circ$. When pane orientation can be tracked by navigational sensors with $\pm 0.1^\circ$ precision, simulated beamplots indicate that resolution, sensitivity, and CTR can be completely recovered.

CHAPTER 6

CONCLUSION AND FUTURE WORK

We have developed a framework for the efficient simulation of fluid-structure coupling for large membrane-type transducer arrays. The framework is based on a boundary element model with computational acceleration provided by a multi-level fast multipole algorithm. This simulation approach has a number of important features which distinguish it from existing numerical and semi-analytical models. First, simulations are not confined by symmetry or periodicity assumptions; arrays can have finite size and arbitrary layouts. Second, membranes can possess arbitrary shapes and properties, such as gap size, thickness, stiffness, and bias. Third, because acoustic interactions are calculated on a node-to-node basis, contributions from anti-symmetric and higher-order membrane modes are captured. Finally, very large arrays with thousands of membranes can be simulated with reasonable computational resources. For a simulation with 230 thousand nodes, computation time averaged 33 min per frequency and memory usage remained below 5 GB.

To expand the applicability of this framework, a hybrid approach was developed to model PMUT arrays. Finite element method simulation of a single PMUT structure provides the necessary information to numerically derive the stiffness matrix and piezoelectrically-induced load. Notably, this approach generalizes to more complicated PMUT geometries featuring small curvature (domed), pre-stressed layers, mass-loading, multiple layers, low aspect ratio (thick), and mixed boundary terminations.

It is envisioned that the simulation tools developed in this work will aid in the design and optimization of realistic imaging arrays. To this end, several examples of practical arrays were demonstrated, including a 32-element CMUT linear array, a 7 by 7 PMUT matrix array, and a 32-element PMUT linear array. These examples were used to investigate cross-talk phenomena and to demonstrate important optimizations such as top electrode coverage and material choice. Additional simulation tools were developed to perform a first-order analysis of imaging performance for a novel foldable large aperture 2-D array for intracardiac echocardiography.

A number of natural extensions to this work have been left to future studies. First, through a mixed-form implementation (e.g., [135, 136]), special formulations of the fast multipole algorithm [137–142], can be used to improve efficiency at low and high frequencies and to avoid low-frequency breakdown due to numerical instability of the translation operator.

Second, several supplemental features may be added to the simulation framework to broaden its applicability. The deposition of thin polymer layers (e.g. PDMS, RTV) on top of arrays have been suggested as a potential method for reducing acoustic cross-talk. It is therefore of great practical interest to be able to simulate these layers, although feasibility of implementation is yet to be investigated. Also important is the simulation of the electrical terminations of the membranes, especially for receive operation, which may be included in the linear system as an additional impedance term.

Finally, by virtue of accelerated matrix-vector products, it is a natural idea to utilize the fast multipole algorithm to perform fast field pressure calculations. Simulated tissue phantoms and beamplots often require the evaluation of millions of points, handled by the summation of the Rayleigh integral. This burdensome calculation can be performed with the FMA in a single matrix-vector product, drastically reducing the necessary computation time.

APPENDIX A

SELECT FMA CODE

Language: Python

Necessary packages: Cython (and C compiler), NumPy, SciPy, H5Py, PyYAML

A.1 Core functions

```
## interaction / complex.pxd
'''
Header for complex datatype.
'''
cdef extern from '<complex.h>' nogil:
    double cabs(double complex)
    double carg(double complex)
    double complex conj(double complex)
    double complex cexp(double complex)
    double creal(double complex)
    double cimag(double complex)
```

```
## interaction / functions.pxd
'''
Header file for 'interaction / functions.pyx'.
'''
cimport numpy as np

cdef double pi = np.pi
cpdef double mag(double[:])
cpdef np.ndarray distance(double[:,:], double[:,:])
cpdef np.ndarray direct_eval(double complex[:], double[:,:], double, double,
    double)
cpdef np.ndarray ff_coeff(double complex[:], double complex[:,:,:])
cpdef np.ndarray calc_exp_part(double[:,:], double[:], double[:,:,:], double)
cpdef np.ndarray nf_eval(double complex[:,:], double complex[:,:,:], double,
    double, double)
cpdef double complex sph_hankel2(int, double)
cpdef np.ndarray ff2nf_op(double, double[:,:], double, int)
cpdef np.ndarray mod_ff2nf_op(double, double[:,:], double, int)
cpdef np.ndarray ff2ff_op(double, double[:,:], double)
cpdef np.ndarray half_fft2(np.ndarray)
cpdef np.ndarray half_ifft2(np.ndarray)
cpdef np.ndarray fft_interpolate(np.ndarray, int, int)
cpdef np.ndarray fft_interpolate_theta(np.ndarray, int)
cpdef np.ndarray fft_filter(double complex[:,:], int, int)
cpdef dict fft_quadrule(int, int)
cpdef np.ndarray bandlimited_abs_sin(int)
```

```
## interaction / functions.pyx
'''
Core functions for fast multipole calculations.
'''
import numpy as np
cimport numpy as np
import scipy as sp

import cython
cimport cython

from libc.math cimport sqrt, cos, sin, exp
```

```
from complex cimport cabs, carg, cexp, creal, cimag, conj
from scipy.fftpack import fft, ifft, fft2, ifft2, fftshift, ifftshift
from scipy.special import eval_legendre, hankel2

################################################################################
## BASIC FUNCTIONS ##
################################################################################

cdef double pi = np.pi

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef double mag(double[:] r):
    '''
    Computes the magnitude of a vector.
    '''
    return sqrt(r[0]*r[0] + r[1]*r[1] + r[2]*r[2])

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray distance(double[:,:] a, double[:,:] b):
    '''
    Calculates the pair-wise distance between two sets of points.
    '''
    cdef int M1 = a.shape[0]
    cdef int N1 = a.shape[1]
    cdef int M2 = b.shape[0]
    cdef int N2 = b.shape[1]
    cdef double[:,:] ret = np.zeros((M1, M2), dtype=np.double)
    cdef double d, tmp
    cdef int i, j, k

    for i in xrange(M1):
        for j in xrange(M2):

            d = 0.0

            for k in xrange(N1):

                tmp = a[i,k] - b[j,k]
                d += tmp * tmp

            ret[i,j] = sqrt(d)

    return np.asarray(ret)

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray dir2coord(kdir):
    '''
    Transforms angular directions from spherical to cartesian.
    '''
    theta = kdir[:,:,0]
    phi = kdir[:,:,1]

    x = np.cos(phi)*np.sin(theta)
    y = np.sin(phi)*np.sin(theta)
    z = np.cos(theta)

    kcoord = np.concatenate((x[...,None], y[...,None], z[...,None]), axis=2)

    return kcoord

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef double complex sph_hankel2(int l, double z):
    '''
    Spherical Hankel function of the second kind of order l and argument z
    calculated from Hankel function (does not handle z=0 case).
    '''
    return sqrt(pi/(2*z))*hankel2(l + 0.5, z)
```

```
@cython.boundscheck(False)
@cython.wraparound(False)
def sph_hankel2_np(l, z):
    '''
    Spherical Hankel function of the second kind of order l and argument z
    calculated from Hankel function (does not handle z=0 case) (numpy version).
    '''
    return sqrt(pi/(2*z))*hankel2(l + 0.5, z)

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray direct_eval(double complex[:] q, double[:,:] dist, double k,
    double rho, double c):
    '''
    Evaluates the field pressure directly using the exact method.
    '''
    cdef M = dist.shape[0]
    cdef N = dist.shape[1]
    cdef double complex[:] ret = np.zeros(M, dtype=np.complex128)
    cdef int i, j
    cdef double complex tmp
    cdef double r

    for i in xrange(M):

        tmp = 0.0j

        for j in xrange(N):

            r = dist[i,j]

            if r == 0.0: continue

            tmp += cexp(-1j*k*r)/r*q[j]

        ret[i] = 1j*k*rho*c/(4*pi)*tmp

    return np.asarray(ret)

################################################################################
## FAST MULTIPOLE FUNCTIONS ##
################################################################################
'''
Implements fast multipole functions based on 'high-frequency' fast multipole
method. A good primer on this method can be found here:

[1] Rahola, Jussi. "Diagonal forms of the translation operators in the fast
multipole algorithm for scattering problems." BIT Numerical Mathematics 36.2
(1996): 333-358.

To make the method compatible with an FFT-based uniform sampling scheme,
a modified translation operator is used which is explained here:

[2] Cecka, Cris, and Eric Darve. "Fourier-based fast multipole method for the
Helmholtz equation." SIAM Journal on Scientific Computing 35.1 (2013): A79-A103.
'''
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray ff_coeff(double complex[:] q, double complex[:,:,:] exp_part):
    '''
    Far-field signature coefficients of a collection of sources in
    the specified far-field directions.
    '''
    cdef int M1 = q.shape[0]
    cdef int M2 = exp_part.shape[1]
    cdef int N2 = exp_part.shape[2]
    cdef double complex[:,:] coeff = np.zeros((M2, N2), dtype=np.complex128)
    cdef double complex tmp1
    cdef int i, j, l

    for i in xrange(M2):
        for j in xrange(N2):
```

```
            tmp1 = 0j

            for l in xrange(M1):
                tmp1 += q[l]*conj(exp_part[l,i,j])

            coeff[i,j] = tmp1

    return np.asarray(coeff)

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray calc_exp_part(double[:,:] nodes, double[:] center,
    double[:,:,:] kcoord, double k):
    '''
    Calculates the exponential part of the evaluation equation (see nf_eval).
    '''
    cdef int M1 = nodes.shape[0]
    cdef int N1 = nodes.shape[1]
    cdef int M2 = kcoord.shape[0]
    cdef int N2 = kcoord.shape[1]
    cdef int O2 = kcoord.shape[2]
    cdef double complex[:,:,:] exp_part = np.zeros((M1, M2, N2),
        dtype=np.complex128)
    cdef double tmp1
    cdef int i, j, l, m

    for i in xrange(M1):
        for j in xrange(M2):
            for l in xrange(N2):

                tmp1 = 0.0

                for m in xrange(N1):
                    tmp1 += (nodes[i,m] - center[m])*kcoord[j,l,m]

                exp_part[i,j,l] = cexp(1j*k*tmp1)

    return np.asarray(exp_part)

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray nf_eval(double complex[:,:] coeff,
    double complex[:,:,:] exp_part, double k, double rho, double c):
    '''
    Evaluate the pressure field at the specified field point(s) using
    near-field signature coefficients.
    '''
    cdef int M1 = coeff.shape[0]
    cdef int N1 = coeff.shape[1]
    cdef int nnodes = exp_part.shape[0]
    cdef double prefactor = k*k*rho*c/(16*pi*pi)
    cdef double complex[:] total = np.zeros(nnodes, dtype=np.complex128)
    cdef int i, j, l
    cdef double complex tmp1

    for i in xrange(nnodes):

        tmp1 = 0j

        for j in xrange(M1):
            for l in xrange(N1):

                tmp1 += coeff[j,l]*exp_part[i,j,l]

        total[i] = prefactor*tmp1

    return np.asarray(total)

@cython.boundscheck(False)
@cython.wraparound(False)
```

```
cpdef np.ndarray ff2nf_op(double r, double[:,:] cos_angle, double k,
    int trans_order):
    '''
    Standard far-field to near-field translation operator (faster version).
    '''
    cdef int M = cos_angle.shape[0]
    cdef int N = cos_angle.shape[1]
    cdef double complex[:,:] ret = np.zeros((M, N), dtype=np.complex128)
    cdef double z = k*r
    cdef double complex tmp
    cdef int l, i, j
    cdef double[:] leg = np.zeros(trans_order + 1, dtype=np.float64)
    cdef double complex[:] sphkl = np.zeros(trans_order + 1,
        dtype=np.complex128)

    sphkl = sph_hankel2_np(np.arange(trans_order + 1), z)

    for i in xrange(M):
        for j in xrange(N):

            tmp = 0j

            leg = eval_legendre(np.arange(trans_order + 1), cos_angle[i, j])

            for l in xrange(trans_order + 1):

                tmp += (2*l + 1)*(1j**l)*sphkl[l]*leg[l]

            ret[i,j] = tmp

    return np.asarray(ret)

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray mod_ff2nf_op(double r, double[:,:] cos_angle, double k,
    int trans_order):
    '''
    Bandlimited modified far-field to near-field translation operator.
    '''
    cdef int kdir_dim1 = cos_angle.shape[0]
    cdef int kdir_dim2 = cos_angle.shape[1]
    cdef int theta_order = (kdir_dim1 - 1)/2
    cdef int phi_order = (kdir_dim2 - 1)
    cdef np.ndarray translation, sinabs, inter1, inter2

    translation = ff2nf_op(r, cos_angle, k, trans_order)
    sinabs = bandlimited_abs_sin(trans_order + theta_order)

    if phi_order > trans_order:
        inter1 = fft_interpolate(translation, trans_order + theta_order,
            phi_order)
    else:
        inter1 = fft_interpolate_theta(translation, trans_order + theta_order)

    inter2 = inter1*sinabs[...,None]

    return fft_filter(0.5*inter2, kdir_dim1, kdir_dim2)

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray bandlimited_abs_sin(int deg):
    '''
    Calculates a bandlimited |sin(theta)| based on its Fourier series.
    '''
    cdef np.ndarray res
    fseries = np.zeros(2*deg + 1, dtype=np.complex128)
    cdef np.ndarray n = np.arange(-deg, deg + 1)

    if deg % 2 == 0:
        fseries[::2] = 2./pi/(1-n[::2]**2)
    else:
```

```
        fseries[1::2] = 2./pi/(1-n[1::2]**2)

    res = fftshift(ifft(ifftshift(np.asarray(fseries))))
    res *= res.size

    return res

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray ff2ff_op(double r, double[:,:] cos_angle, double k):
    '''
    Far-field to far-field shift operator.
    '''
    cdef int M = cos_angle.shape[0]
    cdef int N = cos_angle.shape[1]
    cdef double complex[:,:] shifter = np.zeros((M,N), dtype=np.complex128)
    cdef int i, j

    for i in xrange(M):
        for j in xrange(N):
            shifter[i,j] = cexp(1j*k*r*cos_angle[i,j])

    return np.asarray(shifter)

################################################################################
# INTERPOLATION AND FILTERING (ANTERPOLATION) FUNCTIONS
################################################################################
'''
Implements FFT-based interpolation and filtering based on the following papers:

[1] Sarvas, Jukka. "Performing interpolation and anterpolation entirely by fast
Fourier transform in the 3-D multilevel fast multipole algorithm." SIAM Journal
on Numerical Analysis 41.6 (2003): 2180-2196.

[2] Cecka, Cris, and Eric Darve. "Fourier-based fast multipole method for the
Helmholtz equation." SIAM Journal on Scientific Computing 35.1 (2013): A79-A103.
'''
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray half_fft2(np.ndarray x):
    '''
    Half FFT using spherical property.
    '''
    cdef int M = x.shape[0]
    cdef int N = x.shape[1]

    v = fftshift(fft(x, axis=0), axes=0)
    dummy = np.flipud(v).copy()
    v = np.concatenate((v, dummy), axis=1)

    w = fftshift(fft(v[:(M-1)/2+1,:], axis=1), axes=1)
    dummy = np.flipud(w[:-1,:]).copy()

    n = (-1)**np.arange(-N, N)
    dummy *= n[None,...]

    return np.concatenate((w, dummy), axis=0)

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray half_ifft2(np.ndarray x):
    '''
    Half IFFT using spherical property.
    '''
    cdef int M = x.shape[0]
    cdef int N = x.shape[1]

    return ifft2(ifftshift(x))[:,:N/2]

@cython.boundscheck(False)
@cython.wraparound(False)
```

```
cpdef np.ndarray fft_interpolate(np.ndarray coeff, int kdir_dim1,
    int kdir_dim2):
    '''
    FFT-based interpolation (memory optimized version).
    '''
    cdef int M1 = coeff.shape[0]
    cdef int N1 = coeff.shape[1]
    cdef int M2 = kdir_dim1
    cdef int N2 = kdir_dim2
    cdef int padM, padN

    padM = (M2 - M1)/2
    padN = (N2 - N1)

    #spectrum1 = half_fft2(coeff)
    spectrum1 = half_fft2(ifftshift(coeff, axes=0))
    spectrum2 = np.pad(spectrum1, ((padM, padM),(padN, padN)),
        mode='constant')

    newcoeff = M2*N2/(<double> M1*N1)*half_ifft2(spectrum2)
    newcoeff = fftshift(newcoeff, axes=0)

    return newcoeff

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray fft_interpolate_theta(np.ndarray coeff, int new_order):
    '''
    FFT-based interpolation for theta direction only.
    '''
    cdef int M1 = coeff.shape[0]
    cdef int N1 = coeff.shape[1]
    cdef int M2 = 2*(new_order) + 1

    padM = (M2 - M1)/2

    v = fftshift(fft(ifftshift(coeff, axes=0), axis=0), axes=0)
    dummy = np.flipud(v).copy()

    spectrum1 = np.concatenate((v, dummy), axis=1)
    spectrum2 = np.pad(spectrum1, ((padM, padM),(0, 0)), mode='constant')

    newcoeff = M2/(<double> M1)*ifft(ifftshift(spectrum2, axes=0),
        axis=0)[:,:N1]
    newcoeff = fftshift(newcoeff, axes=0)

    return newcoeff

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef np.ndarray fft_filter(double complex[:,:] coeff, int kdir_dim1,
    int kdir_dim2):
    '''
    FFT-based filtering (memory optimized version).
    '''
    cdef int M1 = coeff.shape[0]
    cdef int N1 = coeff.shape[1]
    cdef int M2 = kdir_dim1
    cdef int N2 = kdir_dim2
    cdef int Mstart, Mstop, Nstart, Nstop

    Mstart = (M1 - M2)/2
    Mstop = Mstart + M2
    Nstart = (N1 - N2)
    Nstop = Nstart + 2*N2

    #spectrum1 = half_fft2(coeff)
    spectrum1 = half_fft2(ifftshift(coeff, axes=0))
    spectrum2 = spectrum1[Mstart:Mstop, Nstart:Nstop]

    newcoeff = M2*N2/(<double> M1*N1)*half_ifft2(spectrum2)
```

```
    newcoeff = fftshift(newcoeff, axes=0)

    return newcoeff

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef dict fft_quadrule(int theta_order, int phi_order):
    '''
    Trapezoidal quadrature rule in theta and phi for integration over a unit
    sphere (memory optimized version).
    '''
    cdef int M, N
    cdef int ntheta, nphi

    # 1: theta/polar
    M = 2*theta_order + 1
    theta = np.linspace(-theta_order*2*np.pi/M, theta_order*2*np.pi/M, M)
    thetaweights = np.ones(M)*2*np.pi/M

    # 2: phi/azimuthal
    N = phi_order + 1
    phi = np.linspace(-np.pi, 0, N, endpoint=False)
    phiweights = np.ones(N)*2*np.pi/N

    weights = thetaweights[:,None].dot(phiweights[None,:])

    ntheta = theta.shape[0]
    nphi = phi.shape[0]

    ktheta, kphi = np.meshgrid(theta, phi, indexing='ij')
    kdir = np.concatenate((ktheta[...,None], kphi[...,None]), axis=2)

    kdir[:(M-1)/2,:,0] *= -1
    kdir[:(M-1)/2,:,1] += pi

    kcoord = dir2coord(kdir)

    quadrule = {}
    quadrule['kdir'] = kdir
    quadrule['kcoord'] = kcoord
    quadrule['weights'] = weights
    quadrule['theta'] = theta
    quadrule['phi'] = phi
    quadrule['theta_order'] = theta_order
    quadrule['phi_order'] = phi_order
    quadrule['ntheta'] = ntheta
    quadrule['nphi'] = nphi
    quadrule['theta_weights'] = thetaweights
    quadrule['phi_weights'] = phiweights

    return quadrule
```

A.2 Data structures

```
## interaction / trees.py
'''
Data structures for the multi-level fast multipole algorithm.
'''
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from itertools import izip
from scipy.interpolate import interp1d
import os.path
import h5py
import sys
from ast import literal_eval
```

```
try:
    from interaction.functions import *
except ImportError:
    raise ImportError('Could not import functions module')

sys.setrecursionlimit(30000)

class Group():
    '''
    Group (Box) has a center, type (root, branch or leaf), uid (unique ID) and
    size; also keeps track of its parent and child Groups.
    '''
    ROOT = 0
    BRANCH = 1
    LEAF = 2
    maxlevel = 1

    def __init__(self, parent, bbox, uid=(0,0,0)):

        # set level
        if parent is None:
            self.level = 0
        else:
            self.level = parent.level + 1

        # set type
        if parent is None:
            self.type = Group.ROOT
        elif self.level == Group.maxlevel:
            self.type = Group.LEAF
        else:
            self.type = Group.BRANCH

        # set uid
        if parent is None:
            self.uid = (0,0,0)
        else:
            self.uid = uid

        # set bbox
        self.bbox = map(float, bbox)

        # set center
        x0, y0, x1, y1 = bbox
        self.center = np.array([x0 + (x1 - x0)/2., y0 + (y1 - y0)/2., 0.0])

        # set nodes
        self.nodes = None

        # set references
        self.parent = parent
        self.children = [None, None, None, None]

    def __repr__(self):

        types = ('Root', 'Branch', 'Leaf')

        return 'Group <uid: (%d, %d, %d), type: %s>' % (self.uid[0],
            self.uid[1], self.uid[2], types[self.type])

    def subdivide(self):
        '''
        Create tree by instantiating child Groups recursively.
        '''
        # termination condition
        if self.type == Group.LEAF:
            return

        # calculate childrens' bbox
        x0, y0, x1, y1 = self.bbox
        xmid, ymid = (x1 - x0)/2., (y1 - y0)/2.
```

```
        bbox00 = x0, y0, x0 + xmid, y0 + ymid
        bbox10 = x0 + xmid, y0, x1, y0 + ymid
        bbox01 = x0, y0 + ymid, x0 + xmid, y1
        bbox11 = x0 + xmid, y0 + ymid, x1, y1

        # calculate childrens' uid
        level, xid, yid = self.uid

        uid00 = level + 1, 2*xid + 0, 2*yid + 0
        uid10 = level + 1, 2*xid + 1, 2*yid + 0
        uid01 = level + 1, 2*xid + 0, 2*yid + 1
        uid11 = level + 1, 2*xid + 1, 2*yid + 1

        # spawn children
        self.children[0] = Group(self, bbox00, uid00)
        self.children[0].subdivide() # << recursion!
        self.children[1] = Group(self, bbox10, uid10)
        self.children[1].subdivide() # << recursion!
        self.children[2] = Group(self, bbox01, uid01)
        self.children[2].subdivide() # << recursion!
        self.children[3] = Group(self, bbox11, uid11)
        self.children[3].subdivide() # << recursion!

    def find(self, uid):
        '''
        Returns the Group with the specified uid, or None if not found.
        '''
        if self.uid == uid:
            return self

        if self.level == uid[0]:
            return None

        for child in [c for c in self.children if c is not None]:

            res = child.find(uid)

            if res is not None:
                return res

        return None

class QuadTree():
    '''
    A QuadTree has up to four children per parent box. Used for 2D problems.
    This is a recursive implementation meant to be simpler and more readable
    (but probably a bit slower).
    '''
    _parameters = ['maxlevel', 'wavenumber', 'node_area', 'density',
        'sound_speed', 'path_to_translation_order_file',
        'path_to_translation_repository']

    def __init__(self, nodes, bounding_box, maxlevel, **kwargs):

        # Read in configuration parameters
        config = dict.fromkeys(QuadTree._parameters)

        for k, v in kwargs.iteritems():
            if k in config:
                config[k] = v

        config['maxlevel'] = maxlevel
        self.config = config

        # Set tree parameters
        Group.maxlevel = maxlevel
        self.bbox = map(float, bounding_box)
        self.apply_counter = 0

        # Plant and grow full tree
        root = Group(parent=None, bbox=bounding_box)
```

```
        root.subdivide()

        # Setup tree and prune
        self.root = root
        self.allgroups = []
        self.leaves = []

        self._traverse(root)
        self._add_nodes(nodes)
        self._prune(root)

        self._find_neighbors(root)

        for group in self.allgroups:
            self._find_ntnn(group)

    def _find(self, uid, group=None):
        '''
        Returns the Group with the specified uid, or None if not found.
        '''
        if group is None:
            group = self.root

        return group.find(uid)

    def _add_nodes(self, nodes):
        '''
        Add nodes to the QuadTree and assigns the nodes to their corresponding
        leaf group.
        '''
        self.nodes = nodes

        x0, y0, x1, y1 = self.root.bbox
        maxid = 2**self.root.maxlevel
        xdim, ydim = (x1 - x0)/maxid, (y1 - y0)/maxid

        # calculate usid of each node: the unique single digit id which
        # identifies the group it belongs to
        xid = np.floor((nodes[:,0] - x0)/xdim).astype(np.intc)
        yid = np.floor((nodes[:,1] - y0)/ydim).astype(np.intc)

        # handle cases where node is on top and right boundaries
        xid[xid == maxid] -= 1
        yid[yid == maxid] -= 1

        usid = xid + maxid*yid

        for group in self.leaves:

            l, xid, yid = group.uid
            group_usid = xid + maxid*yid

            group.node_ids = np.nonzero(usid == group_usid)[0]

            if group.node_ids.size > 0:
                group.nodes = nodes[group.node_ids, :]

    def _prune(self, group):
        '''
        Prunes the QuadTree by removing leaf Groups with no nodes and their
        corresponding branches.
        '''
        if group.type == Group.LEAF:

            if group.nodes is None:

                self.allgroups.remove(group)
                self.leaves.remove(group)

                return True

            return False
```

```
        res = []
        for idx, child in enumerate(group.children):
            if child is not None:

                if self._prune(child): # << recursion!

                    group.children[idx] = None
                    res.append(True)
                else:

                    res.append(False)
            else:
                res.append(True)

        if all(res):

            self.allgroups.remove(group)
            return True

        return False

    def _traverse(self, group):
        '''
        Traverses the tree in order to add all Groups in the tree to a master
        list (called allgroups).
        '''
        self.allgroups.append(group)

        if group.type == Group.LEAF:
            self.leaves.append(group)

        for child in group.children:
            if child is not None:
                self._traverse(child) # << recursion!

    def _find_neighbors(self, group):
        '''
        Has each group in the tree find its touching neighbors.
        '''
        if group.type == Group.ROOT:
            group.neighbors = []

        if group.level > 0:

            level, xid, yid = group.uid
            maxid = 2**level

            # set uid search range
            istart = max(xid - 1, 0)
            istop = min(xid + 1, maxid)
            jstart = max(yid - 1, 0)
            jstop = min(yid + 1, maxid)

            group.neighbors = []

            if level == 1:

                for i in xrange(istart, istop + 1):
                    for j in xrange(jstart, jstop + 1):

                        # skip if uid is itself
                        if i == xid and j == yid:
                            continue

                        res = self._find((level, i, j), group=self.root)

                        if res is not None:
                            group.neighbors.append(res)

            else:
```

```
            parent = group.parent

            for i in xrange(istart, istop + 1):
                for j in xrange(jstart, jstop + 1):

                    # skip if uid is itself
                    if i == xid and j == yid:
                        continue

                    for neighbor in (parent.neighbors + [parent,]):
                        res = self._find((level, i, j), group=neighbor)

                        if res is not None:
                            group.neighbors.append(res)
                            break

    for child in group.children:
        if child is not None:
            self._find_neighbors(child) # << recursion!

def _find_ntnn(self, group):
    '''
    Find the non-touching nearest neighbors for the specified Group
    '''
    parent = group.parent

    group.ntnn = []

    if parent is None:
        return

    for neighbor in parent.neighbors:
        for child in neighbor.children:
            if child is not None:
                if child not in group.neighbors:
                    group.ntnn.append(child)
def setup(self):
    '''
    Set the QuadTree up for solving. This function calls the individual
    setup functions in the correct order.
    '''
    leaves = self.leaves
    config = self.config

    for k, v in config.iteritems():
        if v is None:
            print k
            raise Exception('One or more configuration parameters not set')

    # zero apply counter
    self.apply_counter = 0

    # setup fmm (quadrature rule etc.)
    self._setup_fmm()

    # setup translators
    self._setup_translators()

    #setup shifters
    self._setup_shifters()

    # precompute distances and exp part for leaves
    for group in leaves:

        self._calc_self_dist(group)
        self._calc_neighbor_dist(group)
        self._calc_exp_part(group)

def _setup_fmm(self):
    '''
    Setup quadrature rules and translation operator order.
    '''
```

```
    config = self.config

    k = config['wavenumber']
    trans_order_filepath = config['path_to_translation_order_file']
    maxlevel = config['maxlevel']

    x0, y0, x1, y1 = self.bbox
    xlength, ylength = x1 - x0, y1 - y0

    self.ldata = {}
    # compute far-field angles for each level
    for l in xrange(2, maxlevel + 1):

        # load translation order file
        with h5py.File(trans_order_filepath, 'r') as root:

            ks = root['wavenumbers'][:]
            orders = root[str(l)]['order'][:]


        # check to make sure trans order is always less than or equal to
        # that for the level above
        orders_interp_func = interp1d(ks, orders)
        order = int(orders_interp_func(k))
        if order % 2 == 0:
            order += 1

        self.ldata[l] = fft_quadrule(order, order)
        self.ldata[l]['trans_order'] = order
        self.ldata[l]['group_dims'] = xlength/(2**l), ylength/(2**l)

def _setup_translators(self):
    '''
    Setup translation operators (precalculated) by loading them from a
    database.
    '''
    config = self.config
    ldata = self.ldata

    k = config['wavenumber']
    maxlevel = config['maxlevel']
    filepath = os.path.normpath(config['path_to_translation_repository'])

    x0, y0, x1, y1 = self.bbox
    xlength, ylength = x1 - x0, y1 - y0

    # load translations for every level
    self.translators = {}

    with h5py.File(filepath, 'r') as root:

        for l in xrange(2, maxlevel + 1):

            cache = {}
            key = '{:0.4f}/{:n}'.format(k, l)

            for vec, trans in root[key].iteritems():
                cache[literal_eval(vec)] = trans[:]

            expanded_cache = {}

            for vec, translation in cache.iteritems():

                try:
                    x, y, z = vec
                except Exception:
                    print vec
                    raise Exception

                ntheta, nphi = translation.shape

                # Quadrant II
```

```
                a = np.flipud(translation)[:,nphi/2:]
                b = translation[:,:nphi/2]
                expanded_cache[(-y, x, z)] = \
                    np.ascontiguousarray(np.concatenate((a, b), axis=1))

                # Quadrant III
                expanded_cache[(-x, -y, z)] = \
                    np.ascontiguousarray(np.flipud(translation))

                # Quadrant IV
                a = translation[:,nphi/2:]
                b = np.flipud(translation)[:,:nphi/2]
                expanded_cache[(y, -x, z)] = \
                    np.ascontiguousarray(np.concatenate((a, b), axis=1))

            cache.update(expanded_cache)

            self.translators[l] = cache

    # assign each group's translators
    allgroups = self.allgroups

    for group in allgroups:

        group.translators = []
        l = group.level

        if l < 2:
            continue

        xdim, ydim = ldata[l]['group_dims']

        for fargroup in group.ntnn:

            rx, ry, _ = group.center - fargroup.center

            x = int(round(rx/xdim))
            y = int(round(ry/ydim))
            z = 0

            group.translators.append(self.translators[l][(x, y, z)])

def _setup_shifters(self):
    '''
    Setup shift operators (calculated here).
    '''
    config = self.config
    ldata = self.ldata

    k = config['wavenumber']
    maxlevel = config['maxlevel']

    self.shifters = {}

    for l in xrange(2, maxlevel + 1):

        xdim, ydim = ldata[l]['group_dims']
        kcoordT = ldata[l]['kcoord'].transpose((0,2,1))
        r = np.sqrt(xdim**2 + ydim**2)/2.

        # define direction unit vectors for the four quadrants
        rhat00 = np.array([1, 1, 0])/np.sqrt(2) # lower left group
        rhat10 = np.array([-1, 1, 0])/np.sqrt(2) # lower right group
        rhat01 = np.array([1, -1, 0])/np.sqrt(2) # upper left group
        rhat11 = np.array([-1, -1, 0])/np.sqrt(2) # upper right group

        # calculate shifters from magnitude and angle
        shift00 = ff2ff_op(r, rhat00.dot(kcoordT), k)
        shift10 = ff2ff_op(r, rhat10.dot(kcoordT), k)
        shift01 = ff2ff_op(r, rhat01.dot(kcoordT), k)
        shift11 = ff2ff_op(r, rhat11.dot(kcoordT), k)
```

```
            self.shifters[l] = []
            self.shifters[l].append(shift00)
            self.shifters[l].append(shift10)
            self.shifters[l].append(shift01)
            self.shifters[l].append(shift11)

    def _calc_exp_part(self, group):
        '''
        Calculate the exponential part.
        '''
        config = self.config
        ldata = self.ldata

        k = config['wavenumber']

        nodes = group.nodes
        center = group.center
        l = group.level
        kcoord = ldata[l]['kcoord']

        group.exp_part = calc_exp_part(nodes, center, kcoord, k)

    def _calc_self_dist(self, group):
        '''
        Calculates distances between nodes in the specified Group.
        '''
        nodes = group.nodes
        group.self_dist = distance(nodes, nodes)

    def _calc_neighbor_dist(self, group):
        '''
        Calculates distances between nodes in the specified Group and nodes in
        neighbor Groups.
        '''
        nodes = group.nodes
        neighbors = group.neighbors
        group.neighbor_dist = []

        for neighbor in neighbors:
            group.neighbor_dist.append(distance(nodes, neighbor.nodes))

    def apply(self, strengths):

        config = self.config
        for v in config.itervalues():
            if v is None:
                raise Exception('One or more configuration parameters not set')

        self.apply_counter += 1

        root = self.root
        leaves = self.leaves
        nnodes = self.nodes.shape[0]

        for group in leaves:
            self._calc_coeffs(group, strengths)

        self._uptree(root)
        self._downtree(root)

        # calculate pressures
        pres = np.zeros(nnodes, dtype=np.complex128)

        for group in self.leaves:
            self._calc_pres(group, strengths, pres)

        return pres

    def _calc_coeffs(self, group, strengths):
        '''
        Calculates the far-field coefficients for the specified Group.
        '''
```

```
        node_ids = group.node_ids
        exp_part = group.exp_part

        q = np.ascontiguousarray(strengths[node_ids])
        group.coeffs = ff_coeff(q, exp_part)

    def _uptree(self, group):
        '''
        Upward traversal of the QuadTree. Starting with Leaf Groups, far-field
        coefficients are shifted, interpolated and collected by the Parent.
        '''
        # skip leaves
        if group.type == Group.LEAF:
            return

        for child in group.children:
            if child is not None:
                self._uptree(child) # << recursion!

        # skip levels 0 and 1
        if group.level < 2:
            return

        shifters = self.shifters

        ntheta1, nphi1 = shifters[group.level + 1][0].shape
        ntheta2, nphi2 = shifters[group.level][0].shape

        sum_coeffs = np.zeros((ntheta1, nphi1), dtype=np.complex128)

        for child, shifter in izip(group.children, shifters[group.level + 1]):
            if child is not None:
                sum_coeffs += child.coeffs*shifter

        if ntheta2 > ntheta1:
            group.coeffs = fft_interpolate(sum_coeffs, ntheta2, nphi2)
        else:
            group.coeffs = sum_coeffs

    def _downtree(self, group):
        '''
        Downward traversal of the QuadTree. Starting at L2, the coefficients of
        non-touching neighbor Groups are translated and aggregated. These
        near-field coefficients are then shifted, filtered, and assigned to
        child Groups.
        '''
        # skip levels 0 and 1
        if group.level > 1:

            if group.level == 2:
                aggr_coeffs = np.zeros_like(group.coeffs, dtype=np.complex128)
            else:
                aggr_coeffs = group.aggr_coeffs

            for fargroup, translator in izip(group.ntnn, group.translators):
                aggr_coeffs += fargroup.coeffs*translator

            # skip this part for leaves
            if group.type != Group.LEAF:

                shifters = self.shifters

                ntheta1, nphi1 = shifters[group.level][0].shape
                ntheta2, nphi2 = shifters[group.level + 1][0].shape

                if ntheta2 < ntheta1:
                    aggr_coeffs = fft_filter(aggr_coeffs, ntheta2, nphi2)

                for child, shifter in izip(group.children,
                    shifters[group.level + 1]):
                    if child is not None:
                        child.aggr_coeffs = np.conj(shifter)*aggr_coeffs
```

```
        for child in group.children:
            if child is not None:
                self._downtree(child) # << recursion!

    def _calc_pres(self, group, strengths, pres):
        '''
        Pressure evaluation. After uptree and downtree traversals, pressure
        is calculated at every node either directly or by evaluation of
        near-field coefficients.
        '''
        config = self.config
        ldata = self.ldata

        k = config['wavenumber']
        maxlevel = config['maxlevel']
        rho = config['density']
        c = config['sound_speed']
        s_n = config['node_area']

        node_ids = group.node_ids
        q = np.ascontiguousarray(strengths[node_ids])
        self_dist = group.self_dist
        weight = ldata[maxlevel]['weights'][0,0]

        # pressure from nodes within group
        pres[node_ids] += direct_eval(q, self_dist, k, rho, c)

        # pressure from neighbor groups
        for neighbor, dist in izip(group.neighbors, group.neighbor_dist):

            q1 = np.ascontiguousarray(strengths[neighbor.node_ids])
            pres[node_ids] += direct_eval(q1, dist, k, rho, c)

        # pressure from far groups
        pres[node_ids] += weight*nf_eval(group.aggr_coeffs, group.exp_part,
            k, rho, c)

        # self pressures (piston radiation)
        a_eff = np.sqrt(s_n/np.pi)
        pres[node_ids] += rho*c*(0.5*(k*a_eff)**2 + 1j*8/(3*np.pi)*k*a_eff)/2. \
            *(q/s_n)
```

A.3 Worst-case tests

```
## interaction / tests.py
'''
Tester functions for fast multipole calculations. These functions are used in
scripts to determine order numbers for the translation operator and quadrature
rules. These represent the 'worst-case' translations.
'''
import numpy as np
import scipy as sp
from matplotlib import pyplot as plt
from scipy.interpolate import interp1d

try:
    from functions import *
except ImportError:
    raise ImportError('Could not import functions module')

def nine_point_test(k, xdim, ydim, level, translation_order, theta_order,
    phi_order, rho, c):
    '''
    Nine point test: (1) sources are placed in a grid around a box (1 center
    source and 8 on the periphery, (2) source field is translated from the
    source box to an evaluation box with center located two box lengths away,
```

```
    (3) translated field is evaluated at 9 points on a grid in the evaluation
    box.
    '''
    Dx, Dy = xdim/2**level, ydim/2**level

    # set source and target box for one-box buffer scheme
    source_origin = np.array([0.0, 0.0, 0.0])
    target_origin = np.array([2*Dx, 0.0, 0.0])

    # locate sources and targets at 9 points in each box
    sources = (np.c_[np.array([0.0, 0.5, 1.0, 0, 0.5, 1.0, 0.0, 0.5, 1.0]),
        np.array([0.0, 0.0, 0.0, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0 ]),
        np.zeros(9)]*Dx - np.array([Dx/2., Dy/2., 0]) + source_origin)
    targets = sources + target_origin

    strength = np.ones(1, dtype=np.complex128)
    quadrule = fft_quadrule(theta_order, phi_order)

    # calculate translation operator
    delta_r = target_origin - source_origin
    rhat = delta_r/mag(delta_r)
    kcoord = quadrule['kcoord']
    weight = quadrule['weights'][0,0]
    cos_angle = rhat.dot(kcoord.transpose((0,2,1)))

    translator = mod_ff2nf_op(mag(delta_r), cos_angle, k, translation_order)
    targets_exp_part = calc_exp_part(targets, target_origin, kcoord, k)

    pres_exact = []
    pres_fmm = []

    for src in sources:

        src = np.atleast_2d(src)

        # FMA calculation
        src_exp_part = calc_exp_part(src, source_origin, kcoord, k)
        coeffs = ff_coeff(strength, src_exp_part)

        pres_fmm.append(weight*nf_eval(coeffs*translator, targets_exp_part, k,
            rho, c))

        # direct calculation
        pres_exact.append(direct_eval(strength, distance(targets, src), k, rho,
            c))

    pres_exact = np.array(pres_exact).ravel()
    pres_fmm = np.array(pres_fmm).ravel()

    return pres_fmm, pres_exact

def calculate_error_measures(pres_fmm, pres_exact):
    '''
    Calculates error measures between exact pressure and FMA-evaluated pressure.
    '''
    # relative amplitude error in percent
    amp_rel_error = ((np.abs(pres_fmm) - np.abs(pres_exact))/
        np.abs(pres_exact))*100
    max_amp_rel_error = np.max(np.abs(amp_rel_error))

    # relative phase error (re 2pi) in percent based on smallest angle
    phase_fmm = np.angle(pres_fmm)
    phase_exact = np.angle(pres_exact)
    phase_error = np.arccos(np.round(np.cos(phase_fmm)*np.cos(phase_exact) +
        np.sin(phase_fmm)*np.sin(phase_exact), 10))
    phase_rel_error = phase_error/(2*np.pi)*100
    max_phase_rel_error = np.max(np.abs(phase_rel_error))

    # amplitude and phase bias (useful for determing occurence of breakdown)
    amp_bias = np.mean(amp_rel_error)/np.std(amp_rel_error)
    phase_bias = np.mean(phase_rel_error)/np.std(phase_rel_error)
```

```
    return max_amp_rel_error, max_phase_rel_error, amp_bias, phase_bias

# verbose
def print_results(max_amp_error, max_phase_error, amp_bias, phase_bias):

    print 'Amplitude error within 0.1% ...', '[', max_amp_error <= 0.1, ']'
    print 'Amplitude error within 1% ...', '[', max_amp_error <= 1, ']'
    print 'Amplitude error within 5% ...', '[', max_amp_error <= 5, ']'
    print 'Amplitude error within 10% ...', '[', max_amp_error <= 10, ']'
    print 'Phase error within 0.1% ...', '[', max_phase_error <= 0.1, ']'
    print 'Phase error within 1% ...', '[', max_phase_error <= 1, ']'
    print 'Phase error within 5% ...', '[', max_phase_error <= 5, ']'
    print 'Phase error within 10% ...', '[', max_phase_error <= 10, ']'
```

A.4 Scripts

```
## interaction / scripts / create_translation_orders_file.py
'''
This script will use the specified tester function to empirically determine the
translation operator order and quadrature order needed to achieve the error
target.

'''
import _path

import numpy as np
import h5py
import yaml
import sys
from multiprocessing import Pool, Lock
from importlib import import_module
import os.path
from os import makedirs
from itertools import product
from tqdm import tqdm

from interaction.tests import calculate_error_measures

def pass_condition(amp_error, phase_error, amp_bias, phase_bias):
    '''
    '''
    if amp_error <= error_target and phase_error <= error_target:
        return True

    return False

def breakdown_condition(amp_errors, phase_errors):
    '''
    '''
    dy = np.diff(amp_errors)
    dyy = np.diff(dy)

    if len(dy) > 5:
        if np.all(dy[-5:] > 10):
            if np.all(dyy[-4:] > 0):
                return True

    return False

def process(args):
    '''
    '''
    l, i = args
    k = wavenumbers[i]
    breakdown, passed = 0, 0
```

```
    stop_order = start_order + search_range
    orders = range(start_order, stop_order + 1, 2)

    amp_errors = np.zeros_like(orders, dtype=np.float64)
    phase_errors = np.zeros_like(orders, dtype=np.float64)

    for j, order in enumerate(orders):

        amp_error, phase_error, amp_bias, phase_bias = \
            calculate_error_measures(*test(k, xdim, ydim, 1, order, order,
            order, density, sound_speed))

        amp_errors[j] = amp_error
        phase_errors[j] = phase_error

        # check pass condition
        if pass_condition(amp_error, phase_error, amp_bias, phase_bias):

            raw_order = order
            passed = 1
            break

        # check breakdown condition
        if breakdown_condition(amp_errors[:j+1], phase_errors[:j+1]):

            raw_order = start_order + np.argmin(amp_errors[:j+1])*2
            breakdown = 1
            break

        if order == stop_order:
            raw_order = start_order + np.argmin(amp_errors[:j+1])*2

    return l, i, raw_order, passed, breakdown

def groom_over_wavenumber(raw_order, breakdown):
    '''
    '''
    order = np.zeros_like(raw_order)

    order[0] = raw_order[0]

    for i in xrange(1, len(raw_order)):

        if breakdown[i]:

            order[i] = raw_order[i]
            continue

        if raw_order[i] < order[i -1]:
            order[i] = order[i - 1]
        else:
            order[i] = raw_order[i]

    return order

def despike(raw_order, breakdown):
    '''
    '''
    order = raw_order.copy()

    dy = np.diff(raw_order)

    for i in xrange(1, len(dy) - 1):

        if dy[i] == 0:
            continue

        if not breakdown[i]:
            if np.sign(dy[i]) == -np.sign(dy[i - 1]):
                if np.abs(dy[i]) >= 2:
                    order[i] = order[i + 1]
```

```
    return order


def groom_over_level():
    '''
    '''
    with h5py.File(filepath, 'r+') as root:

        for l in xrange(maxlevel - 1, minlevel - 1, -1):

            order1 = root[str(l) + '/' + 'order'][:]
            breakdown1 = root[str(l) + '/' + 'breakdown'][:]
            order2 = root[str(l + 1) + '/' + 'order'][:]

            for i in xrange(len(order1)):
                if not breakdown1[i]:
                    if order1[i] < order2[i]:
                            order1[i] = order2[i]

            root[str(l) + '/' + 'order'][:] = order1

def write_to_file(l, results):
    '''
    '''
    with h5py.File(filepath, 'r+') as root:

        key = str(l) + '/' + 'raw_order'
        root.create_dataset(key, data=results['raw_order'], compression='gzip',
            compression_opts=9)

        key = str(l) + '/' + 'order'
        root.create_dataset(key, data=results['order'], compression='gzip',
            compression_opts=9)

        key = str(l) + '/' + 'breakdown'
        root.create_dataset(key, data=results['breakdown'], compression='gzip',
            compression_opts=9)

        key = str(l) + '/' + 'passed'
        root.create_dataset(key, data=results['passed'], compression='gzip',
            compression_opts=9)

def make_filepath(folder, xdim, ydim, error_target):
    '''
    '''
    str_format = 'translation_order_xdim_{:0.4f}m_ydim_{:0.4f}m_tol_{:0.1f}.h5'
    path = os.path.join(folder, str_format.format(xdim, ydim, error_target))

    return path

################################### GLOBAL ###################################

# Read in YAML configuration file
config = yaml.load(open(sys.argv[1], 'r'))

# Read in configuration parameters
number_of_threads = config['number_of_threads']
f_fine_start = config['f_fine_start']
f_fine_stop = config['f_fine_stop']
f_fine_step = config['f_fine_step']
f_coarse_start = config['f_coarse_start']
f_coarse_stop = config['f_coarse_stop']
f_coarse_step = config['f_coarse_step']
sound_speed = config['sound_speed']
density = config['density']
save_folder = os.path.join(_path.packagedir,
    os.path.normpath(config['save_folder']))
xdim, ydim = config['xdim'], config['ydim']
minlevel = config['minlevel']
maxlevel = config['maxlevel']
error_target = config['error_target']
```

```
test_module = config['test_module']
test_function = config['test_function']
start_order = 1
search_range = 500

# Create other global vars
filepath = make_filepath(save_folder, xdim, ydim, error_target)
lock = Lock()

# Determine frequencies/wavenumbers for calculation
freqs = np.concatenate((np.arange(f_fine_start, f_fine_stop, f_fine_step),
    np.arange(f_coarse_start, f_coarse_stop + f_coarse_step, f_coarse_step)))
wavenumbers = 2*np.pi*freqs/sound_speed

# Determine levels for calculation
levels = range(minlevel, maxlevel + 1)

# import test module and function
tests_module = import_module(test_module)
test = getattr(tests_module, test_function)

################################################################################

if __name__ == '__main__':

    # Check for existing file and existing wavenumbers
    if os.path.isfile(filepath):
        raise Exception('File already exists')

    else:
        # Make directories if they do not exist
        dirpath = os.path.dirname(filepath)
        if not os.path.exists(dirpath):
            makedirs(dirpath)

        # Create hdf5 file
        with h5py.File(filepath, 'w') as root:
            root.create_dataset('wavenumbers', data=wavenumbers,
                compression='gzip', compression_opts=9)

    try:
        # Start multiprocessing pool and run process
        pool = Pool(max(number_of_threads, maxlevel - 1))
        result = pool.imap_unordered(process, product(levels,
            xrange(len(wavenumbers))))

        data = {}
        for l in levels:

            data[l] = {}
            data[l]['raw_order'] = np.zeros_like(wavenumbers)
            data[l]['breakdown'] = np.zeros_like(wavenumbers)
            data[l]['passed'] = np.zeros_like(wavenumbers)
            data[l]['done'] = np.zeros_like(wavenumbers)

        for l, i, ro, psd, bd in tqdm(result, desc='Progress',
            total=len(levels)*len(wavenumbers)):

            data[l]['raw_order'][i] = ro
            data[l]['breakdown'][i] = bd
            data[l]['passed'][i] = psd
            data[l]['done'][i] = 1

            if np.all(data[l]['done']):

                arg1 = despike(data[l]['raw_order'], data[l]['breakdown'])
                arg2 = data[l]['breakdown']
                data[l]['order'] = groom_over_wavenumber(arg1, arg2)

                write_to_file(l, data[l])
```

```
        if minlevel != maxlevel:
            groom_over_level()

    except Exception as e:
        print e

    finally:
        pool.terminate()
        pool.close()
```

```
## interaction / scripts / create_translation_repository.py
'''
This script will pre-calculate the translation operators for a given bounding
box, max level, and frequency steps for a multi-level fast multipole algorithm.
This can take hours to days depending on the number of threads available, size
of bounding box, number of levels etc. The output is stored in a single H5 file.

To use, run with a corresponding yaml config file for setting the input
parameters.

python create_translation_repository.py <path to config file>
'''
import _path

import numpy as np
import h5py
import yaml
import sys
from multiprocessing import Pool, Lock
from scipy.interpolate import interp1d
import os.path
from os import makedirs
from tqdm import tqdm

try:
    from interaction.functions import *
except ImportError:
    raise ImportError('Could not import functions module')

def create_ldata(k):

    ldata = {}

    for l in levels:

        trans_order = int(orders_interp_func[l](k))
        if trans_order % 2 == 0:
            trans_order += 1

        data = fft_quadrule(trans_order, trans_order)
        data['trans_order'] = trans_order
        data['group_dims'] = xdim/(2**l), ydim/(2**l)

        ldata[l] = data

    return ldata

def calculate_translators(k, ldata):

    maxlevel = config['maxlevel']

    def mag(r):
        return np.sqrt(np.sum(r**2))

    translators = {}

    for l in xrange(2, maxlevel + 1):

        kcoordT = ldata[l]['kcoord'].transpose((0,2,1))
        trans_order = ldata[l]['trans_order']
        xdim, ydim = ldata[l]['group_dims']
```

```
        x, y, z = np.mgrid[1:4,0:4,0:1:1j]
        uniques = np.c_[x.ravel(), y.ravel(), z.ravel()].astype(int)
        uniques = uniques[2:,:]

        cache = {}

        for vec in uniques:

            r = vec*np.array([xdim, ydim, 1])
            rhat = r/mag(r)

            cos_angle = rhat.dot(kcoordT)

            cache[tuple(vec)] = np.ascontiguousarray(mod_ff2nf_op(mag(r),
                cos_angle, k, trans_order))

        translators[l] = cache

    return translators

def write_to_file(k, translators):

    with h5py.File(filepath, 'r+') as root:
        for l, val in translators.iteritems():
            for vec, trans in val.iteritems():

                key = make_h5_key(k, l ,vec)
                root.create_dataset(key, data=trans, compression='gzip',
                    compression_opts=9)

def process(k):

    ldata = create_ldata(k)

    translators = calculate_translators(k, ldata)

    with lock:
        write_to_file(k, translators)

def make_h5_key(k, l, vec ):

    str_format = '{:0.4f}/{:n}/{}'
    key = str_format.format(k, l, str(vec))

    return key

def make_filepath(folder, xdim, ydim):

    str_format = 'translators_xdim_{:0.4f}m_ydim_{:0.4f}m.h5'
    path = os.path.join(folder, str_format.format(xdim, ydim))

    return path

################################### GLOBAL ###################################

# Read in YAML configuration file
config = yaml.load(open(sys.argv[1], 'r'))

# Read in configuration parameters
number_of_threads = config['number_of_threads']
f_start = config['f_start']
f_stop = config['f_stop']
f_step = config['f_step']
sound_speed = config['sound_speed']
#overwrite = config['overwrite']
save_folder = os.path.join(_path.packagedir,
    os.path.normpath(config['save_folder']))
xdim, ydim = config['xdim'], config['ydim']
minlevel = config['minlevel']
maxlevel = config['maxlevel']
```

```
translation_order_filepath = os.path.join(_path.packagedir,
    os.path.normpath(config['path_to_translation_order_file']))

# Create other global vars
filepath = make_filepath(save_folder, xdim, ydim)
levels = range(minlevel, maxlevel + 1)
lock = Lock()

# Determine frequencies/wavenumbers for calculation
f = np.arange(f_start, f_stop + f_step, f_step)
k = 2*np.pi*f/sound_speed

# Create interpolation function for translation orders
orders_interp_func = {}
with h5py.File(translation_order_filepath, 'r') as root:

    ks = root['wavenumbers'][:]

    for l in levels:

        orders = root[str(l) + '/' + 'order'][:]
        orders_interp_func[l] = interp1d(ks, orders)

################################################################################

if __name__ == '__main__':

    # Check for existing file and existing wavenumbers
    if os.path.isfile(filepath):

        #if not overwrite:

        with h5py.File(filepath, 'r') as root:
            existing_k = [float(x) for x in root.iterkeys()]

        # Set to iterate over only new wavenumbers
        new_k = np.array([x for x in k if round(x, 4) not in existing_k])

    else:

        # Make directories if they do not exist
        dirpath = os.path.dirname(filepath)
        if not os.path.exists(dirpath):
            makedirs(dirpath)

        # Create hdf5 file
        with h5py.File(filepath, 'w') as root:
            pass

        # Set to iterate over all wavenumbers
        new_k = k

    try:

        # Start multiprocessing pool and run process
        pool = Pool(number_of_threads)
        result = pool.imap_unordered(process, new_k)

        for r in tqdm(result, desc='Progress', total=len(new_k)):
            pass

    except Exception as e:
        print e

    finally:
        pool.terminate()
        pool.close()
```

REFERENCES

[1] N. Bom, C. Lancee, and F. Van Egmond, "An ultrasonic intracardiac scanner," *Ultrasonics*, no. March, pp. 72–76, 1972.

[2] Z. Hijazi, Z. Wang, Q. Cao, P. Koenig, D. Waight, and R. Lang, "Transcatheter closure of atrial septal defects and patent foramen ovale under intracardiac echocardiographic guidance: feasibility and comparison with transesophageal echocardiography.," *Catheterization and Cardiovascular Interventions*, vol. 52, no. 2, pp. 194–9, 2001.

[3] J. E. Banchs, P. Patel, G. V. Naccarelli, and M. D. Gonzalez, "Intracardiac echocardiography in complex cardiac catheter ablation procedures," *Journal of Interventional Cardiac Electrophysiology*, vol. 28, no. 3, pp. 167–184, 2010.

[4] Siemens Healthcare. `http://www.healthcare.siemens.com/ultrasound/cardiovascular/acuson-sc2000-ultrasound-system/use`.

[5] Diagnostic and Interventional Cardiology. `http://www.dicardiology.com/article/ultrasound-sees-increasing-use-interventional-procedures`.

[6] W. J. Toulis, "Flexural-extensional electro-mechanical transducer," Sept. 20 1966. US Patent 3,274,537.

[7] G. Lockwood and F. S. Foster, "Optimizing sparse two-dimensional transducer arrays using an effective aperture approach," in *Ultrasonics Symposium, 1994. Proceedings., 1994 IEEE*, vol. 3, pp. 1497–1501, 1994.

[8] L. Landini, R. Sarnelli, E. Picano, and M. Salvadori, "Evaluation of frequency dependence of backscatter coefficient in normal and atherosclerotic aortic walls," *Ultrasound in Medicine and Biology*, vol. 12, pp. 397–401, May 1986.

[9] B. I. Raju and M. A. Srinivasan, "High-frequency ultrasonic attenuation and backscatter coefficients of in vivo normal human dermis and subcutaneous fat," *Ultrasound in Medicine and Biology*, vol. 27, pp. 1543–1556, Nov. 2001.

[10] M. O'Donnell, J. W. Mimbs, and J. G. Miller, "Relationship between collagen and ultrasonic backscatter in myocardial tissue," *Journal of the Acoustical Society of America*, vol. 69, pp. 580–588, Feb. 1981.

[11] K. K. Shung, R. Sigelmann, and J. Reid, "Scattering of ultrasound by blood," *IEEE Transactions on Biomedical Engineering*, vol. BME-23, pp. 460–467, Nov. 1976.

[12] E. J. Benjamin, M. J. Blaha, S. E. Chiuve, M. Cushman, S. R. Das, R. Deo, S. D. de Ferranti, J. Floyd, M. Fornage, C. Gillespie, *et al.*, "Heart disease and stroke statistics2017 update: a report from the american heart association," *Circulation*, vol. 135, no. 10, pp. e146–e603, 2017.

[13] D. P. Faxon and D. O. Williams, "The changing face of interventional cardiology," 2012.

[14] D. Auerbach, J. Maeda, and C. Steiner, "Hospital stays with cardiac stents, 2009: Statistical brief #128," 2006.

[15] D. R. Holmes, R. A. Nishimura, F. L. Grover, R. G. Brindis, J. D. Carroll, F. H. Edwards, E. D. Peterson, J. S. Rumsfeld, D. M. Shahian, V. H. Thourani, *et al.*, "Annual outcomes with transcatheter valve therapy: from the sts/acc tvt registry," *Journal of the American College of Cardiology*, vol. 66, no. 25, pp. 2813–2823, 2015.

[16] Z. M. Hijazi, K. Shivkumar, and D. J. Sahn, "Intracardiac Echocardiography During Interventional and Electrophysiological Cardiac Catheterization," *Circulation*, vol. 119, no. 4, pp. 587–596, 2009.

[17] T. Cieszynski, "Intracardiac method for the investigation of structure of the heart with the aid of ultrasonics," *Archivum Immunologiae et Therapiae Experimentalis*, vol. 8, p. 551, 1960.

[18] N. Bom, H. Ten Hoff, C. T. Lancée, W. J. Gussenhoven, and J. G. Bosch, "Early and recent intraluminal ultrasound devices.," *International Journal of Cardiac Imaging*, vol. 4, no. 2-4, pp. 79–88, 1989.

[19] R. Omoto, "Intracardiac scanning of the heart with the aid of ultrasonic intravenous probe," *Japanese Heart Journal*, vol. 8, no. 6, pp. 569–581, 1967.

[20] R. Eggleton, C. Townsend, G. Kossoff, J. Herrick, R. Hunt, G. Templeton, and J. Mitchell, "Computerised ultrasonic visualization of dynamic ventricular configurations. 8th ICMBE," *Palmer House, Chicago IL*, pp. 10–3, 1969.

[21] P. Koenig and Q. L. Cao, "Echocardiographic guidance of transcatheter closure of atrial septal defects: is intracardiac echocardiography better than transesophageal echocardiography?," *Pediatric Cardiology*, vol. 26, no. 2, pp. 135–9, 2005.

[22] S. G. Dravid, B. Hope, and J. J. McKinnie, "Intracardiac echocardiography in electrophysiology: a review of current applications in practice.," *Echocardiography*, vol. 25, no. 10, pp. 1172–5, 2008.

[23] J. M. Cooper and L. M. Epstein, "Use of Intracardiac Echocardiography to Guide Ablation of Atrial Fibrillation," *Circulation*, vol. 104, pp. 3010–3013, 2001.

[24] X. Cheng, D. Lemmerhirt, O. Kripfgans, M. Zhang, C. Yang, C. Rich, and J. Fowlkes, "CMUT-in-CMOS ultrasonic transducer arrays with on-chip electronics," in *International Solid-State Sensors, Actuators and Microsystems Conference, 2009.*, pp. 1222–1225, 2009.

[25] P.-C. Eccardt, K. Niederer, T. Scheiter, and C. Hierold, "Surface micromachined ultrasound transducers in CMOS technology," in *Ultrasonics Symposium, 1996. Proceedings., 1996 IEEE*, vol. 2, pp. 959–962, IEEE, 1996.

[26] P. C. Eccardt and K. Niederer, "Micromachined ultrasound transducers with improved coupling factors from a CMOS compatible process," *Ultrasonics*, vol. 38, no. 1, pp. 774–780, 2000.

[27] J. Zahorian, M. Hochman, T. Xu, S. Satir, G. Gurun, M. Karaman, and F. L. Degertekin, "Monolithic CMUT-on-CMOS integration for intravascular ultrasound applications.," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 58, no. 12, pp. 2659–67, 2011.

[28] I. O. Wygant, X. Zhuang, D. T. Yeh, Ö. Oralkan, A. S. Ergun, M. Karaman, and B. T. Khuri-Yakub, "Integration of 2D cmut arrays with front-end electronics for volumetric ultrasound imaging," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 55, no. 2, pp. 327–342, 2008.

[29] R. J. Przybyla, S. E. Shelton, A. Guedes, I. I. Izyumin, M. H. Kline, D. A. Horsley, and B. E. Boser, "In-air rangefinding with an aln piezoelectric micromachined ultrasound transducer," *Sensors Journal, IEEE*, vol. 11, no. 11, pp. 2690–2697, 2011.

[30] I. O. Wygant, M. Kupnik, J. C. Windsor, W. M. Wright, M. S. Wochner, G. G. Yaralioglu, M. F. Hamilton, and B. T. Khuri-Yakub, "50 kHz capacitive micromachined ultrasonic transducers for generation of highly directional sound with parametric arrays," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 56, no. 1, pp. 193–203, 2009.

[31] Y. Lu, H. Tang, S. Fung, Q. Wang, J. Tsai, M. Daneman, B. Boser, and D. Horsley, "Ultrasonic fingerprint sensor using a piezoelectric micromachined ultrasonic transducer array integrated with complementary metal oxide semiconductor electronics," *Applied Physics Letters*, vol. 106, no. 26, p. 263503, 2015.

[32] D. T. Yeh, S. Member, I. O. Wygant, M. O. Donnell, B. T. Khuri-yakub, O. Oralkan, I. O. Wygant, M. O'Donnell, and B. T. Khuri-yakub, "3-D ultrasound imaging using a forward-looking CMUT ring array for intravascular / intracardiac applications," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 53, no. 6, pp. 1202–1211, 2006.

[33] C. Tekes, J. Zahorian, T. Xu, M. W. Rashid, S. Satir, G. Gurun, M. Karaman, J. Hasler, and F. L. Degertekin, "CMUT-based volumetric ultrasonic imaging array design for forward looking ICE and IVUS applications," in *Conf Proc IEEE Eng Med Biol Soc*, vol. 8675, p. 86750B, Mar. 2013.

[34] D. E. Dausch, J. B. Castellucci, D. R. Chou, and O. T. Von Ramm, "Theory and operation of 2-D array piezoelectric micromachined ultrasound transducers," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 55, no. 11, pp. 2484–2492, 2008.

[35] W. Liao, W. Liu, J. Rogers, F. Usmani, Y. Tang, B. Wang, H. Jiang, and H. Xie, "Piezoelectric micromachined ultrasound tranducer array for photoacoustic imaging," in *Solid-State Sensors, Actuators and Microsystems (TRANSDUCERS & EUROSENSORS XXVII), 2013 Transducers & Eurosensors XXVII: The 17th International Conference on*, pp. 1831–1834, 2013.

[36] S. Lani, K. G. Sabra, and F. L. Degertekin, "Super-resolution ultrasonic imaging of stiffness variations on a microscale active metasurface," *Applied Physics Letters*, vol. 108, no. 8, p. 084104, 2016.

[37] H. C. Hayes, "Sound generating and directing apparatus," Dec. 22 1936. US Patent 2,064,911.

[38] K. D. Rolt, "History of the flextensional electroacoustic transducer," *Journal of the Acoustical Society of America*, vol. 87, no. 3, pp. 1340–1349, 1990.

[39] R. Newnham, J. Zhang, and R. Meyer Jr, "Cymbal transducers: a review," in *2000 12th IEEE Int. Symp. Applications Ferroelectrics. (ISAF 2000)*, vol. 1, pp. 29–32, 2000.

[40] B. Houston, J. Bucaro, T. Yoder, L. Kraus, J. Tressler, J. Fernandez, T. Montgomery, and T. Howarth, "Broadband low frequency sonar for non-imaging based identification," in *OCEANS'02 MTS/IEEE*, vol. 1, pp. 383–387, IEEE, 2002.

[41] J. Marchal and P. Cervenka, "Feasibility of b-scan imaging in sediment by means of parametric transmission technique," *Acta Acustica united with Acustica*, vol. 90, no. 1, pp. 62–69, 2004.

[42] D. Brady and J. C. Preisig, "Underwater acoustic communications," *Wireless Communications: Signal Processing Perspectives*, vol. 8, pp. 330–379, 1998.

[43] E. Maione, K. K. Shung, R. J. Meyer Jr, J. W. Hughes, R. E. Newnham, and N. B. Smith, "Transducer design for a portable ultrasound enhanced transdermal drug-delivery system," *Ultrasonics, Ferroelectrics, and Frequency Control, IEEE Transactions on*, vol. 49, no. 10, pp. 1430–1436, 2002.

[44] J. Luis, E.-J. Park, R. J. Meyer, and N. B. Smith, "9F-4 rectangular cymbal arrays for ultrasonic transdermal insulin delivery," in *2007 IEEE Int. Ultrason. Symp.*, pp. 840–843, 2007.

[45] S. J. Kavros and E. C. Schenck, "Use of noncontact low-frequency ultrasound in the treatment of chronic foot and leg ulcerations: a 51-patient analysis," *Journal of the American Podiatric Medical Association*, vol. 97, no. 2, pp. 95–101, 2007.

[46] A. S. Savoia, B. Mauti, and G. Caliano, "A low frequency broadband flextensional ultrasonic transducer array," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 63, no. 1, pp. 128–138, 2016.

[47] T. Eriksson, S. Ramadas, A. Unger, M. Hoffman, M. Kupnik, and S. Dixon, "Flexural transducer arrays for industrial non-contact applications," in *Ultrasonics Symposium (IUS), 2015 IEEE International*, pp. 1–4, 2015.

[48] Y. Huang, A. S. Ergun, E. Hæggström, M. H. Badi, and B. T. Khuri-Yakub, "Fabricating capacitive micromachined ultrasonic transducers with wafer-bonding technology," *Journal of Microelectromechanical Systems*, vol. 12, no. 2, pp. 128–137, 2003.

[49] J. Knight, J. McLean, and F. L. Degertekin, "Low temperature fabrication of immersion capacitive micromachined ultrasonic transducers on silicon and dielectric substrates," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 51, no. 10, pp. 1324–1333, 2004.

[50] Y. Qiu, J. V. Gigliotti, M. Wallace, F. Griggio, C. E. Demore, S. Cochran, and S. Trolier-McKinstry, "Piezoelectric micromachined ultrasound transducer (pmut) arrays for integrated sensing, actuation and imaging," *Sensors*, vol. 15, no. 4, pp. 8020–8041, 2015.

[51] F. Akasheh, T. Myers, J. D. Fraser, S. Bose, and A. Bandyopadhyay, "Development of piezoelectric micromachined ultrasonic transducers," *Sensors and Actuators, A: Physical*, vol. 111, no. 2, pp. 275–287, 2004.

[52] X. Jin, O. Oralkan, F. L. Degertekin, and B. T. Khuri-Yakub, "Characterization of one-dimensional capacitive micromachined ultrasonic immersion transducer arrays," vol. 48, no. 3, pp. 750–760.

[53] A. Caronti, D. Fiasca, G. Caliano, M. Pappalardo, and E. Cianci, "Experimental study of acoustic coupling in cmut arrays by optical interferometry," in *Ultrasonics, 2003 IEEE Symposium on*, vol. 2, pp. 1960–1969, 2003.

[54] B. Bayram, M. Kupnik, G. G. Yaralioglu, O. Oralkan, A. S. Ergun, D. S. Lin, S. H. Wong, and B. T. Khuri-Yakub, "Finite element modeling and experimental characterization of crosstalk in 1-D CMUT arrays," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 54, no. 2, pp. 418–429, 2007.

[55] M. Wilm, A. Reinhardt, V. Laude, R. Armati, W. Daniau, and S. Ballandras, "Three-dimensional modelling of micromachined-ultrasonic-transducer arrays operating in water," *Ultrasonics*, vol. 43, no. 6, pp. 457–465, 2005.

[56] S. Lani, K. G. Sabra, and F. L. Degertekin, "Modal and transient analysis of membrane acoustic metasurfaces," *Journal of Applied Physics*, vol. 117, no. 4, p. 045308, 2015.

[57] S. Ballandras, M. Wilm, W. Daniau, a. Reinhardt, V. Laude, and R. Armati, "Periodic finite element/boundary element modeling of capacitive micromachined ultrasonic transducers," *Journal of Applied Physics*, vol. 97, no. 3, 2005.

[58] A. Caronti, A. Savoia, G. Caliano, and M. Pappalardo, "Acoustic coupling in capacitive microfabricated ultrasonic transducers: modeling and experiments.," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 52, no. 12, pp. 2220–2234, 2005.

[59] D. T. Porter, "Self-and mutual-radiation impedance and beam patterns for flexural disks in a rigid plane," *The Journal of the Acoustical Society of America*, vol. 36, no. 6, pp. 1154–1161, 1964.

[60] M. N. Senlik, S. Olcum, H. Köymen, and A. Atalar, "Radiation impedance of an array of circular capacitive micromachined ultrasonic transducers," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 57, no. 4, pp. 969–976, 2010.

[61] H. Oguz, A. Atalar, and H. Koymen, "Equivalent circuit-based analysis of cmut cell dynamics in arrays," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 60, no. 5, pp. 1016–1024, 2013.

[62] A. Atalar, H. Koymen, and H. K. Oguz, "Rayleigh-Bloch waves in CMUT arrays," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 61, no. 12, pp. 2139–2148, 2014.

[63] S. Akhbari, F. Sammoura, and L. Lin, "Equivalent circuit models for large arrays of curved and flat piezoelectric micromachined ultrasonic transducers," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 63, no. 3, pp. 432–447, 2016.

[64] A. Rønnekleiv, "CMUT array modeling through free acoustic CMUT modes and analysis of the fluid CMUT interface through Fourier transform methods.," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 52, no. 12, pp. 2173–84, 2005.

[65] D. Certon, F. Teston, and F. Patat, "A finite difference model for cMUT devices.," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 52, no. 12, pp. 2199–2210, 2005.

[66] C. Meynier, F. Teston, and D. Certon, "A multiscale model for array of capacitive micromachined ultrasonic transducers.," *Journal of the Acoustical Society of America*, vol. 128, no. 5, pp. 2549–2561, 2010.

[67] S. Satir and F. L. Degertekin, "A nonlinear lumped model for ultrasound systems using cmut arrays," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 62, pp. 1865–1879, Oct. 2015.

[68] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, 1987.

[69] J. Song, C.-C. Lu, and W. C. Chew, "Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects," *IEEE Transactions on Antennas and Propagation*, vol. 45, no. 10, pp. 1488–1493, 1997.

[70] Y. Ohno, R. Yokota, H. Koyama, G. Morimoto, a. Hasegawa, G. Masumoto, N. Okimoto, Y. Hirano, H. Ibeid, T. Narumi, and M. Taiji, "Petascale molecular dynamics simulation using the fast multipole method on K computer," *Computer Physics Communications*, vol. 185, no. 10, pp. 2575–2585, 2014.

[71] M. S. Tong, W. C. Chew, and M. J. White, "Multilevel fast multipole algorithm for acoustic wave scattering by truncated ground with trenches.," *Journal of the Acoustical Society of America*, vol. 123, no. 5, pp. 2513–2521, 2008.

[72] H. Wu, Y. Liu, and W. Jiang, "A fast multipole boundary element method for 3D multi-domain acoustic scattering problems based on the BurtonMiller formulation," *Engineering Analysis with Boundary Elements*, vol. 36, no. 5, pp. 779–788, 2012.

[73] D. R. Wilkes and A. J. Duncan, "Acoustic coupled fluid-structure interactions using a unified fast multipole boundary element method," *Journal of the Acoustical Society of America*, vol. 137, no. 4, pp. 2158–2167, 2015.

[74] J. Zahorian, S. Satir, and F. L. Degertekin, "Analytical-finite element hybrid model for cmut arrays with arbitrary membrane geometry," in *Ultrasonics Symposium (IUS), 2012 IEEE International*, pp. 584–587, 2012.

[75] S. Satir, J. Zahorian, and F. L. Degertekin, "A large-signal model for CMUT arrays with arbitrary membrane geometry operating in non-collapsed mode," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 60, pp. 2426–2439, Nov. 2013.

[76] N. Senegond, A. Boulme, C. Plag, F. Teston, and D. Certon, "Fast time-domain modeling of fluid-coupled cmut cells: from the single cell to the 1-D linear array element," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 60, no. 7, pp. 1505–1518, 2013.

[77] P.-C. Eccardt, P. Wagner, and S. Hansen, "5f-1 analytical models for micromachined transducers-an overview," in *Ultrasonics Symposium, 2006. IEEE*, pp. 572–581, 2006.

[78] L. E. Kinsler, A. R. Frey, A. B. Coppens, and J. V. Sanders, *Fundamentals of Acoustics*. John Wiley & Sons, Inc., 4th ed., 2000.

[79] Y. Saad and M. H. Schultz, "GMres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[80] C. F. Gerald and P. O. Wheatley, *Applied Numerical Analysis*. Pearson Education, 7th ed., 2004.

[81] R. Coifman, V. Rokhlin, and S. Wandzura, "Fast multiple method for the wave equation: a pedestrian prescription," *IEEE Antennas and Propagation Magazine*, vol. 35, no. 3, pp. 7–12, 1993.

[82] R. Yokota, H. Ibeid, and D. Keyes, "Fast multipole method as a matrix-free hierarchical low-rank approximation," *arXiv preprint arXiv:1602.02244*, 2016.

[83] J. Rahola, "Diagonal forms of the translation operators in the fast multipole algorithm for scattering problems," *BIT*, vol. 36, no. 2, pp. 333–358, 1996.

[84] J. Sarvas, "Performing interpolation and anterpolation entirely by fast Fourier transform in the 3-D multilevel fast multipole algorithm," *SIAM Journal on Numerical Analysis*, vol. 41, no. 6, pp. 2180–2196, 2003.

[85] C. Cecka and E. Darve, "Fourier-based fast multipole method for the helmholtz equation," *SIAM Journal on Scientific Computing*, vol. 35, no. 1, pp. A79–A103, 2013.

[86] S. Ohnuki and W. C. Chew, "Numerical analysis of local interpolation error for 2D-MLFMA," *Microwave and Optical Technology Letters*, vol. 36, no. 1, pp. 8–12, 2003.

[87] R. Jakob-Chien and B. K. Alpert, "A Fast Spherical Filter with Uniform Resolution," *Journal of Computational Physics*, vol. 136, no. 2, pp. 580–584, 1997.

[88] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, vol. 55. Courier Corporation, 1964.

[89] E. Darve, "The fast multipole method: numerical implementation," *Journal of Computational Physics*, vol. 160, no. 1, pp. 195–240, 2000.

[90] T. Szabo, *Diagnostic Ultrasound Imaging: Inside Out*. Biomedical Engineering, Elsevier Science, 2013.

[91] Python Software Foundation. `http://www.python.org`.

[92] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python." `http://www.scipy.org/`, 2001–.

[93] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[94] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2011.

[95] A. H. Baker, E. R. Jessup, and T. Manteuffel, "A technique for accelerating the convergence of restarted GMRES," *SIAM Journal on Matrix Aanalysis and Applications*, vol. 26, no. 4, pp. 962–984, 2005.

[96] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK users' guide*. Society for Industrial and Applied Mathematics, third ed., 1999.

[97] Q. Zhu, T. Chen, H. Liu, L. Sun, T. Wang, C. Lee, X. Le, and J. Xie, "An aln-based piezoelectric micro-machined ultrasonic transducer (pmut) array," in *Nanotechnology (IEEE-NANO), 2016 IEEE 16th International Conference on*, pp. 731–734, 2016.

[98] D. E. Dausch, K. H. Gilchrist, J. B. Carlson, S. D. Hall, J. B. Castellucci, and O. T. von Ramm, "In vivo real-time 3-d intracardiac echo using pmut arrays," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 61, no. 10, pp. 1754–1764, 2014.

[99] S. Dong, K. Uchino, L. Li, and D. Viehland, "Analytical solutions for the transverse deflection of a piezoelectric circular axisymmetric unimorph actuator," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 54, no. 6, pp. 1240–1249, 2007.

[100] G. Percin and B. T. Khuri-Yakub, "Piezoelectrically actuated flextensional micromachined ultrasound transducers. i. theory," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 49, no. 5, pp. 573–584, 2002.

[101] F. Sammoura, S. Akhbari, and L. Lin, "An analytical solution for curved piezoelectric micromachined ultrasonic transducers with spherically shaped diaphragms," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 61, no. 9, pp. 1533–1544, 2014.

[102] F. Sammoura and S.-G. Kim, "Theoretical modeling and equivalent electric circuit of a bimorph piezoelectric micromachined ultrasonic transducer," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 59, no. 5, 2012.

[103] H. Choi, M. Anderson, J. Ding, and A. Bandyopadhyay, "A two-dimensional electromechanical composite plate model for piezoelectric micromachined ultrasonic transducers (pmuts)," *Journal of Micromechanics and Microengineering*, vol. 20, no. 1, p. 015013, 2009.

[104] K. M. Smyth, *Design and modeling of a PZT thin film based piezoelectric micromachined ultrasonic transducer (PMUT)*. PhD thesis, Massachusetts Institute of Technology, 2012.

[105] K. Smyth, S. Bathurst, F. Sammoura, and S.-G. Kim, "Analytic solution for n-electrode actuated piezoelectric disk with application to piezoelectric micromachined ultrasonic transducers," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 60, no. 8, pp. 1756–1767, 2013.

[106] F. Sammoura, K. Smyth, S.-G. Kim, and L. Lin, "An accurate equivalent circuit for the clamped circular multiple-electrode pmut with residual stress," in *Ultrasonics Symposium (IUS), 2013 IEEE International*, pp. 275–278, 2013.

[107] F. Akasheh, J. D. Fraser, S. Bose, and A. Bandyopadhyay, "Piezoelectric micromachined ultrasonic transducers: Modeling the influence of structural parameters on device performance," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 52, no. 3, pp. 455–468, 2005.

[108] B. Shieh, K. G. Sabra, and F. L. Degertekin, "Efficient broadband simulation of fluid-structure coupling for membrane-type acoustic transducer arrays using the multilevel fast multipole algorithm," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 63, no. 11, pp. 1967–1979, 2016.

[109] H. Lee, J. Tak, W. Moon, and G. Lim, "Effects of mutual impedance on the radiation characteristics of transducer arrays," *Journal of the Acoustical Society of America*, vol. 115, no. 2, pp. 666–679, 2004.

[110] A. Boulmé and D. Certon, "Design of broadband linear micromachined ultrasonic transducer arrays by means of boundary element method coupled with normal mode theory," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 62, no. 9, pp. 1704–1716, 2015.

[111] J. G. Gualtieri, J. A. Kosinski, and A. Ballato, "Piezoelectric materials for acoustic wave applications," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 41, no. 1, pp. 53–59, 1994.

[112] K. Tsubouchi, K. Sugai, and N. Mikoshiba, "Aln material constants evaluation and saw properties on aln/al 2 o 3 and aln/si," in *1981 Ultrasonics Symposium*, pp. 375–380, 1981.

[113] G. Carlotti, G. Socino, A. Petri, and E. Verona, "Acoustic investigation of the elastic properties of zno films," *Applied Physics Letters*, vol. 51, no. 23, pp. 1889–1891, 1987.

[114] N. Ledermann, P. Muralt, J. Baborowski, S. Gentil, K. Mukati, M. Cantoni, A. Seifert, and N. Setter, "{1 0 0}-textured, piezoelectric pb (zr x, ti 1- x) o 3 thin films for mems: integration, deposition and properties," *Sensors and Actuators, A: Physical*, vol. 105, no. 2, pp. 162–170, 2003.

[115] L. Lian and N. Sottos, "Effects of thickness on the piezoelectric and dielectric properties of lead zirconate titanate thin films," *Journal of Applied Physics*, vol. 87, no. 8, pp. 3941–3949, 2000.

[116] F. Sammoura, K. Smyth, and S.-G. Kim, "Optimizing the electrode size of circular bimorph plates with different boundary conditions for maximum deflection of piezoelectric micromachined ultrasonic transducers," *Ultrasonics*, vol. 53, no. 2, pp. 328–334, 2013.

[117] M. Karaman and M. O'Donnell, "Subaperture processing for ultrasonic imaging," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 45, no. 1, pp. 126–135, 1998.

[118] T. M. Carpenter, M. W. Rashid, M. Ghovanloo, D. M. Cowell, S. Freear, and F. L. Degertekin, "Direct digital demultiplexing of analog tdm signals for cable reduction in ultrasound imaging catheters," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 63, no. 8, pp. 1078–1085, 2016.

[119] D. Powell and G. Hayward, "Flexible ultrasonic transducer arrays for nondestructive evaluation applications. ii. performance assessment of different array configurations," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 43, no. 3, pp. 393–402, 1996.

[120] O. Roy, S. Mahaut, and O. Casula, "Control of the ultrasonic beam transmitted through an irregular profile using a smart flexible transducer: modelling and application," *Ultrasonics*, vol. 40, no. 1, pp. 243–246, 2002.

[121] R. J. McGough, A. Owens, D. Cindric, J. Heim, and T. Samulski, "The fabrication of conformal ultrasound phased arrays for thermal therapy," in *Engineering in Medicine and Biology Society, 2000. Proceedings of the 22nd Annual International Conference of the IEEE*, vol. 3, pp. 1617–1620, 2000.

[122] R. J. McGough, D. Cindric, and T. V. Samulski, "Shape calibration of a conformal ultrasound therapy array," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 48, no. 2, pp. 494–505, 2001.

[123] R. S. Singh, M. O. Culjat, S. P. Vampola, K. Williams, Z. D. Taylor, and H. Lee, "P3d-6 simulation, fabrication, and characterization of a novel flexible, conformal ultrasound transducer array," in *Ultrasonics Symposium, 2007. IEEE*, pp. 1824–1827, 2007.

[124] R. S. Singh, M. Culjat, M. Lee, D. Bennett, S. Natarjan, B. Cox, E. Brown, W. Grundfest, and H. Lee, *Conformal Ultrasound Imaging System*, pp. 211–222. Springer Netherlands, 2011.

[125] P.-C. Li and M. O'Donnell, "Phase aberration correction on two-dimensional conformal arrays," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 42, no. 1, pp. 73–82, 1995.

[126] G. R. Lockwood, P.-C. Li, M. O'Donnell, and F. S. Foster, "Optimizing the radiation pattern of sparse periodic linear arrays," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 43, no. 1, pp. 7–14, 1996.

[127] S. S. Brunke and G. R. Lockwood, "Broad-bandwidth radiation patterns of sparse two-dimensional vernier arrays," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 44, no. 5, pp. 1101–1109, 1997.

[128] G. R. Lockwood, J. R. Talman, and S. S. Brunke, "Real-time 3-d ultrasound imaging using sparse synthetic aperture beamforming," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 45, no. 4, pp. 980–988, 1998.

[129] G. Gurun, M. Hochman, P. Hasler, and F. L. Degertekin, "Thermal-mechanical-noise-based cmut characterization and sensing," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 59, no. 6, 2012.

[130] G. E. Tupholme, "Generation of acoustic pulses by baffled plane pistons," *Mathematika*, vol. 16, no. 02, pp. 209–224, 1969.

[131] P. R. Stepanishen, "Pulsed transmit/receive response of ultrasonic piezoelectric transducers," *Journal of the Acoustical Society of America*, vol. 69, pp. 1815–1827, June 1981.

[132] J. A. Jensen, "Field: A program for simulating ultrasound systems," in *10th NordicBaltic Conf. Biomed. Imaging, vol. 4, supplement 1, part 1:351–353*, 1996.

[133] R. Wagner, S. Smith, J. Sandrik, and H. Lopez, "Statistics of speckle in ultrasound b-scans," *IEEE Transactions on Sonics and Ultrasonics*, vol. 30, pp. 156–163, May 1983.

[134] F. A. Duck, *Physical properties of tissues: a comprehensive reference book*. Academic press, 2013.

[135] L. J. Jiang and W. C. Chew, "A mixed-form fast multipole algorithm," *IEEE Transactions on Antennas and Propagation*, vol. 53, no. 12, pp. 4145–4156, 2005.

[136] N. A. Gumerov and R. Duraiswami, "A broadband fast multipole accelerated boundary element method for the three dimensional Helmholtz equation.," *Journal of the Acoustical Society of America*, vol. 125, no. 1, pp. 191–205, 2009.

[137] R. J. Burkholder and D.-H. Kwon, "High-frequency asymptotic acceleration of the fast multipole method," *Radio Science*, vol. 31, no. 5, pp. 1199–1206, 1996.

[138] L. Greengard, J. Huang, V. Rokhlin, and S. Wandzura, "Accelerating fast multipole methods for the helmholtz equation at low frequencies," *IEEE Computational Science and Engineering*, vol. 5, no. 3, pp. 32–38, 1998.

[139] L. Jiang and W. C. Chew, "Low-frequency fast inhomogeneous plane-wave algorithm (lf-fipwa)," *Microwave and Optical Technology Letters*, vol. 40, no. 2, pp. 117–122, 2004.

[140] E. Darve and P. Havé, "Efficient fast multipole method for low-frequency scattering," *Journal of Computational Physics*, vol. 197, no. 1, pp. 341–363, 2004.

[141] I. Bogaert and F. Olyslager, "A low frequency stable plane wave addition theorem," *Journal of Computational Physics*, vol. 228, no. 4, pp. 1000–1016, 2009.

[142] O. Ergül and B. Karaosmanoglu, "Low-frequency fast multipole method based on multiple-precision arithmetic," *IEEE Antennas and Wireless Propagation Letters*, vol. 13, pp. 975–978, 2014.